

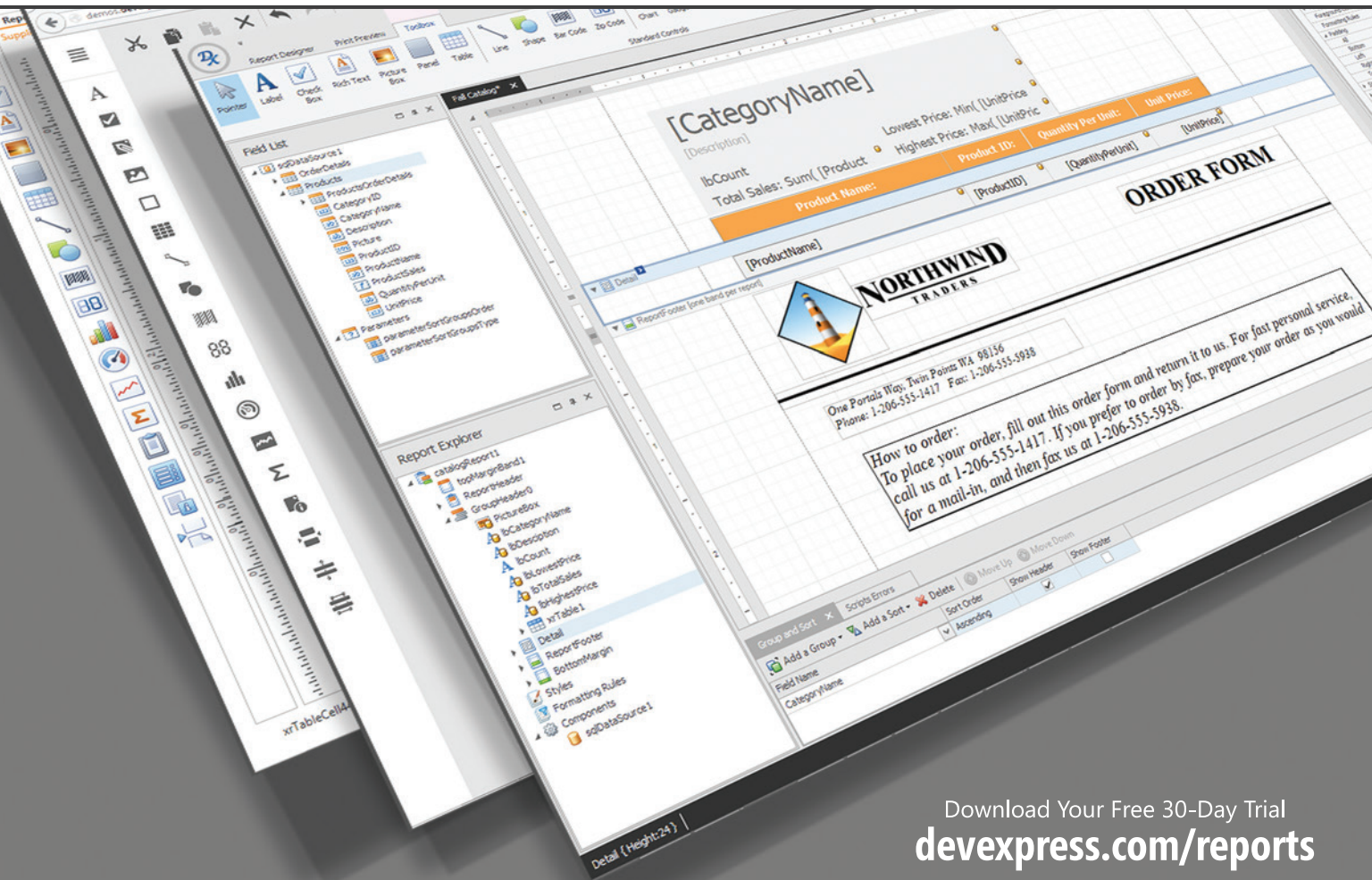
msdn magazine



The Future of Software
Development....18, 28, 38, 48

The Easier Way to Create Reports

A Report Platform optimized for
WPF, ASP.NET and Windows Forms developers.



Download Your Free 30-Day Trial
devexpress.com/reports



"A must have for any serious developer."

Ron Lindsey

**"DevExpress works as a major
workforce multiplier."**

Peter Van Zyl

**"One of the most powerful, feature rich
control suites on the market."**

Chris Todd

#UseTheBest

msdn

magazine



The Future of Software
Development....18, 28, 38, 48

Seeing the World with Xamarin and Microsoft Computer Vision APIs Alessandro Del Sole	18
Capture and Analyze Brain Waves with Azure IoT Hub Benjamin Perkins	28
Solving Business Problems with the Microsoft Bot Framework Srikantan Sankaran	38
Introduction to the HoloLens Adam Tuliper	48
Hidden Disposables Artak Mkrtchyan	56

COLUMNS

FIRST WORD

Any Developer, Any App,
Any Platform
Julia Liuson, page 6

CUTTING EDGE

Code First and
Database Initialization
Dino Esposito, page 8

DATA POINTS

CQRS and EF Data Models
Julie Lerman, page 12

TEST RUN

Solving Sudoku Using
Combinatorial Evolution
James McCaffrey, page 60

THE WORKING PROGRAMMER

How To Be MEAN: Taking a Gulp
Ted Neward, page 68

MODERN APPS

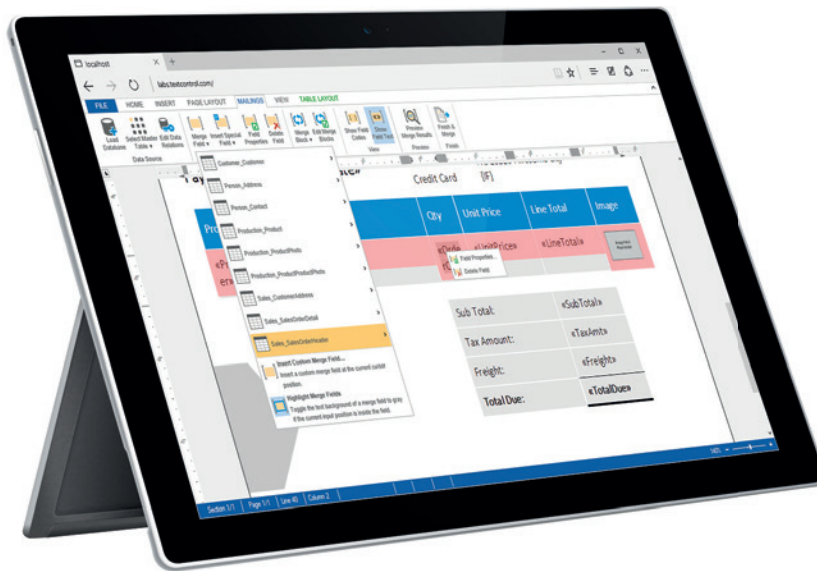
Add Facial Recognition Features
to Your App
Frank La Vigne, page 74

DON'T GET ME STARTED

Brain Droppings
David Platt, page 80

ASP.NET MVC REPORTING

The first cross-browser, true WYSIWYG HTML5-based document editor.
The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

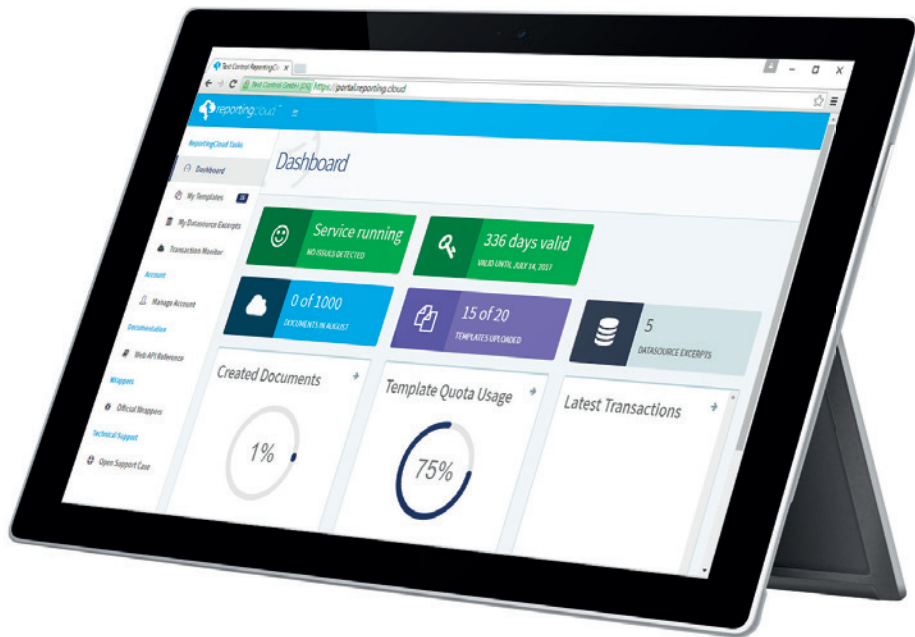


Give your users an MS Word compatible editor to create powerful documents and reporting templates anywhere - in any browser. Feature-complete including mail merge, sections, headers and footers, drawings and shapes, tables, barcodes and charts.

Live demo and 30-day trial version download at:
www.textcontrol.com/reporting

CREATE DOCUMENTS IN THE

Web API powered reporting platform to create
MS Word compatible reports in the cloud.



Use the Text Control ReportingCloud REST web service to create powerful,
MS Word compatible reports with JSON data from all clients such as
.NET, Java, JavaScript, Ruby, PHP, Python, Android, iOS and many more.
Templates can be created and edited directly in any browser.

Free trial account and wrapper documentation at:
www.reporting.cloud

msdn magazine

NOVEMBER 2016 VOLUME 31 NUMBER 11

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Mohammad Al-Sabt mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Features Editor Ed Zintel

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

Chief Marketing Officer
Carmel McDonagh

ART STAFF

Creative Director Jeffrey Langkau

Associate Creative Director Scott Rovin

Senior Art Director Deirdre Hoffman

Art Director Michele Singh

Assistant Art Director Dragutin Cvijanovic

Graphic Designer Erin Horlacher

Senior Graphic Designer Alan Tao

Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca

Regional Sales Manager Christopher Kourtoglou

Account Executive Caroline Stover

Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel

Senior Site Producer, News Kurt Mackie

Senior Site Producer Gladys Rama

Site Producer Chris Paoli

Site Producer, News David Ramel

Director, Site Administration Shane Lee

Site Administrator Biswarup Bhattacharjee

Front-End Developer Anya Smolinski

Junior Front-End Developer Casey Rysavy

Executive Producer, New Media Michael Domingo

Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund

Senior Director, Audience Development

& Data Procurement Annette Levee

Director, Audience Development

& Lead Generation Marketing Irene Fincher

Director, Client Services & Webinar

Production Tracy Cook

Director, Lead Generation Marketing Eric Yoshizuru

Director, Custom Assets & Client Services Mallory Bundy

Senior Program Manager, Client Services

& Webinar Production Chris Flack

Project Manager, Lead Generation Marketing

Mahal Ramos

Coordinator, Lead Generation Marketing

Obum Ukabam

MARKETING

Chief Marketing Officer Carmel McDonagh

Vice President, Marketing Emily Jacobs

Senior Manager, Marketing Christopher Morales

Marketing Coordinator Alicia Chew

Marketing & Editorial Assistant Dana Friedman

ENTERPRISE COMPUTING GROUP EVENTS

Vice President, Events Brent Sutton

Senior Director, Operations Sara Ross

Senior Director, Event Marketing Merikay Marzoni

Events Sponsorship Sales Danna Vedder

Senior Manager, Events Danielle Potts

Coordinator, Event Marketing Michelle Cheng

Coordinator, Event Marketing Chantelle Wallace



Chief Executive Officer

Rajeev Kapur

Chief Operating Officer

Henry Allain

Chief Technology Officer

Erik A. Lindgren

Executive Vice President

Michael J. Valenti

Chairman of the Board

Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 www.1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International Phone: 212-221-9595. E-mail: 1105reprints@parsintl.com. www.magreprints.com/QuickQuote.asp

LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301; E-mail: jljong@meritdirect.com; Web: www.meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. - 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. - 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.





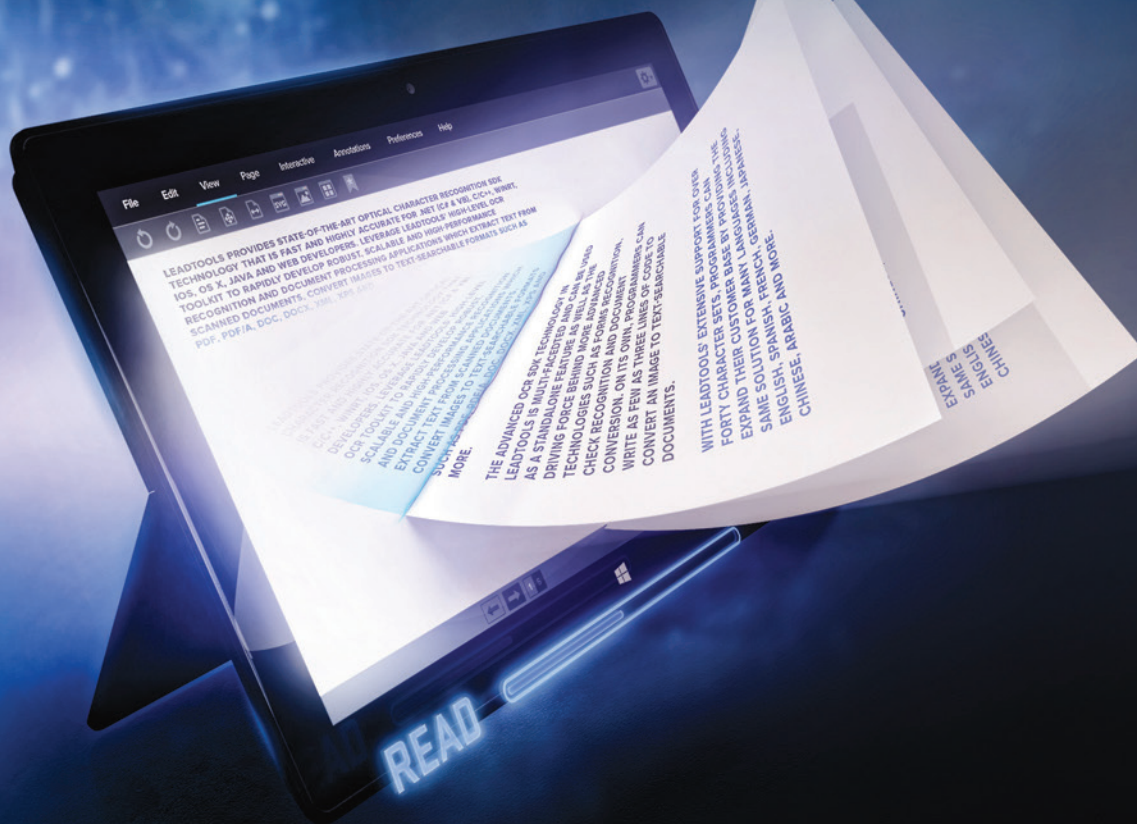
DESKTOP



TABLET



MOBILE



OCR FOR ANY CLIENT, DEVICE OR SERVER. DONE.

LEADTOOLS OCR SDK

Fast, accurate and reliable optical character recognition for use in any application or environment

Utilize multiple cores for unparalleled performance

Supports multiple text recognition engines including OCR, ICR, MICR, MRZ and MRP

Automatically detect, segment and recognize multiple languages on the same document

.NET Windows API Java WinRT Linux iOS OS X Android JavaScript





We Come from the Future

The future ain't what it used to be. The dreams we had of flying cars, moon bases and robot servants may not have panned out (at least, not yet), but we've seen our share of miracles. Like a world-spanning network that delivers instant knowledge, communication and entertainment, and ubiquitous handheld computers that let us reach anything digital—anywhere—with the tap of a finger.

In this issue of *MSDN Magazine*, we explore the tools, technologies and techniques that enable developers to build our next future. From augmented reality environments that mingle the real and the digital, to intelligent services that replace hundreds of brittle apps with contextualized data flows embedded within familiar UIs, to advanced software that interprets facial expressions, drives natural-language interfaces and analyzes brain activity. The sky's the limit, and this month's issue provides a timely glimpse of how to get there.

Things kick off with a feature article that dives into Microsoft Cognitive Services, the family of intelligence and knowledge APIs that, to quote Microsoft, “allow systems to see, hear, speak, understand and interpret needs using natural methods of communication.” Frequent *MSDN Magazine* author Alessandro Del Sole shows how these services can be used to recognize facial characteristics and expressions, either from a digital photo or a smartphone camera input, within cross-platform apps based on Xamarin.Forms.

Del Sole says Cognitive Services opens the stage to an incredible range of development scenarios. He notes as an example the Seeing AI project (bit.ly/2ddzhgQ), which provides machine-generated verbal guidance to visually impaired users, based on visual inputs from wearable cameras, smartphones and other devices. The system can describe surroundings, read out documents, and even interpret and describe the facial expressions of those nearby.

“Cognitive Services are exciting because they offer an opportunity to describe the world in an auto-generated, natural-language description that's available on any platform, on any device, and basically to any development environment supporting REST,” Del Sole says. “This opens the field to building apps that offer experiences tailored for the feelings and needs of customers at specific moments in their lives.”

Also on display are three intriguing features. Benjamin Perkins shows how the Emotiv brain-computer interface can capture brain waves and present them for analysis on the Azure IoT Hub via Stream Analytics. Microsoft HoloLens takes center stage as Adam Tuliper explores the three primary ways developers can set

Microsoft HoloLens and Windows Holographic take center stage as Adam Tuliper explores the three primary ways developers can set up interaction in augmented reality applications—gaze, gesture and voice.

up interaction in augmented reality applications—gaze, gesture and voice. And don't miss Srikantan Sankaran's first in a two-part exploration of development for the new Microsoft Bot Framework. The framework promises to free information and services locked up within piecemeal mobile apps to enable data access and interaction wherever users are—whether it's SMS, Skype, Slack, Facebook Messenger, Office 365 mail or other popular services.

Emerging technologies like the Bot Framework, HoloLens and Cognitive Services are heralding a sea change in software development, as intelligent services and adaptable, transformative UIs begin to redefine the way people engage and interact with software. Today, the fascinating work of enabling those interactions is just beginning.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2016 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.



Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

NEW
v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module



Complete Suite of Accurate PDF Components

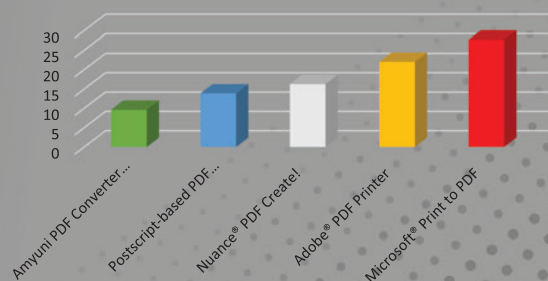
- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment



High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need



AMYUNI 
Technologies

USA and Canada

Toll Free: 1866 926 9864

Support: 514 868 9227

sales@amyuni.com

Europe

UK: 0800-015-4682

Germany: 0800-183-0923

France: 0800-911-248

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.

All development tools available at

www.amyuni.com

Any Developer, Any App, Any Platform

Every company is a software company. Regardless of what industry you're in—farming and agriculture, banking and finance, education and others—technology is transforming how the world runs. Software will be disruptive in multiple dimensions as every company is looking for innovative ways to engage their customers, empower their employees, optimize their operations and transform their products. As a developer, this leads to unique and new opportunities. It's going to be an amazing year for building software.

The core of our vision is “Any Developer, Any App, Any Platform.” With our Visual Studio family, including the recent addition of Xamarin, we are committed to bringing you the most powerful and productive development tools and services to build mobile-first and cloud-first apps across Windows, iOS, Android and Linux.

Our commitment to
cross-platform goes beyond
just the apps you write.

Even a year ago, I couldn't imagine where we'd be. .NET Core, .NET Compiler “Roslyn,” ASP.NET, Visual Studio Code, Xamarin and TypeScript are all open source projects. GitHub just published its 2016 statistics (octoverse.github.com) and Microsoft is the No. 1 leader in open source contributors, above Facebook, Docker and Google, with 16,419 unique contributors. Visual Studio Code (@code) is in the top 10 repositories on GitHub with the most open source contributors with more than 5,855 contributors. I am proud, and humbled, to see how active our open source community is and how much our developer division has led the way in Microsoft embracing open source.

It is also a great time to be a .NET developer! The .NET ecosystem is thriving. With the acquisition of Xamarin, you can now write native C# apps for any mobile platform—iOS, Android or Windows—in any version of Visual Studio, leading to a higher percentage of code sharing across your apps. And with the release of .NET Core 1.0 in June, we've brought you a cross-platform, open source, and modular .NET platform that's designed to help you target the needs of modern applications—highly distributed apps, componentization with microservices, and isolation with containers. Our customers have shipped production solutions with .NET Core and are seeing huge productivity and performance wins:

- NetEase, a leading Internet and gaming company in China, chose Unity and .NET Core for its back-end and front-end services to enable code sharing. When compared to the company's previous Java back-end architecture: “.NET Core has reduced our release cycle by 20 percent and cost on engineering resources by 30 percent.” When speaking about the throughput improvements and cost savings: “Additionally, it has made it possible to reduce the number of virtual machines needed in production by half.”
- Illyriad Games, the team behind “Age of Ascent,” reported a 10-fold increase in performance (bit.ly/2cQq1KN) using ASP.NET Core with Azure Service Fabric.
- Our lab runs (bit.ly/2dpiwT6) using industry benchmarks for Web platforms on Linux, including the TechEmpower Benchmarks (bit.ly/2cQR9eD), show that ASP.NET Core is eight times faster than Node.js, and almost three times better than Go, on the same hardware. We're hoping to see official numbers from TechEmpower soon!

We've designed the .NET standard library to be common across the full .NET Framework, .NET Core, Unity and Xamarin, so any app you create will be able to share common capabilities in the future and you'll be able to reuse your skills across the entire .NET family.

Our commitment to cross-platform goes beyond just the apps you write. Visual Studio Code, hitting 1.0 in April, brings you a lightweight code editor that is fully open sourced and runs everywhere, including Windows, Linux (Debian, Ubuntu, Red Hat, Fedora, CentOS) and OS X. The Visual Studio Code community is incredibly vibrant and growing! We've seen 1.4 million downloads in the last six months alone, and extensions for Visual Studio Code have grown from less than 400 to more than 1,400 in the last year.

This month, at our Connect(); conference in November, we have the opportunity to talk about the next wave of the Microsoft developer platform and services, and how our tools and services will bring a connected end-to-end story from client tools, to DevOps, to an intelligent cloud. I'm excited to share more around how we enable you to be productive across any platform, and the commitment to openness in how we work. ■

JULIA LIUSON is the corporate vice president of the Visual Studio and .NET Framework teams at Microsoft. She is responsible for developer tools and services, including the programming languages and runtimes designed for a broad base of software developers and development teams, as well as for the Visual Studio, Visual Studio Code, and the .NET Framework lines of products and services.

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

All trademarks or registered trademarks are property of their respective owners.



Code First and Database Initialization

Though the term “DevOps” is relatively new and recently has grown to encompass many more activities—most notably automated testing and deployment—I believe the first example of an automatable developer operation is as old as the software itself. I’m referring to the specific ability to create and initialize the database during the application setup. Many software companies develop vertical systems that are then sold to a variety of customers and adapted to their needs.

The aspects that can be customized depend on the characteristics of the product and the business domain, but I dare say that at the very minimum any vertical software application needs to have a customer-specific database. Therefore, the database must be created with the tables and schemas required by the context and populated with ad hoc data.

Not all required tasks can always be automated and built into the product itself. Imagine, for example, the import of existing data. Whether the data to import resides in Excel files or legacy databases, odds are that an importer must be built in some way to process the data and load it into the new storage. However, if you employed Entity Framework 6.x Code First in the data access layer of the application, then at least the creation of the database schemas

and tables is easy to automate and can transparently happen the first time the application is run.

In this column, I’ll summarize some features that have been available in Code First since the early days from the perspective of a multi-customer application. In particular, I’ll focus on how to create and fill a database and how to programmatically define its name and connection string.

Code First offers both a range of attributes and fluent syntax to define a mapping between class properties and columns of the underlying table.

Laying the Ground of Table Schemas

Let’s assume you have a brand-new Visual Studio project already linked to the Entity Framework 6.x NuGet package. The next step you might want to take is to create a data-access class library or at least a distinct folder in the current project to save all files that in some way concur with the working of the data-access functionality. According to Entity Framework rules, you need to have a `DbContext` class that represents the entry point in the application’s data management subsystem. Here’s an example of such an application-specific class:

```
public class RegionsContext : DbContext
{
    ...
}
```

To the application’s eyes, the `DbContext`-derived class is no more and no less than the database. The class is expected to expose a handful of `DbSet<T>` properties, one for each collection of entities being managed by the application. A `DbSet<T>` property is logically equivalent to a table in a physical database:

```
public class RegionsContext : DbContext
{
    public DbSet<Region> Regions { get; set; }
}
```

The net effect of the code snippet here is to enable the application to work with a relational database that contains a table named `Regions`. What about the schema of the table? The schema is determined by the public layout of the `Region` class. Code First offers both a range of

Figure 1 Sample Web.config File as Modified to Support Entity Framework Code First

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="entityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,..." />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>

  <connectionStrings>
    <!-- Your explicit connection strings -->
  </connectionStrings>

  <entityFramework>
    <defaultConnectionFactory type=
      "System.Data.Entity.Infrastructure.SqlConnectionFactory,
      EntityFramework">
    <parameters>
      <parameter value="Data Source=(local); Integrated Security=True;" />
    </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient"
        type="System.Data.Entity.SqlServer.SqlProviderServices, ..." />
    </providers>
  </entityFramework>
</configuration>
```

BUSINESS FILE APIS

TRY RISK FREE - 30 Day Trial

Open, Create, Convert, Print and Save files from your apps!



Aspose.Total

Aspose.Cells

XLS, CSV, PDF, SVG, HTML, PNG
BMP, XPS, JPG, SpreadsheetML...

Aspose.Words

DOC, RTF, PDF, HTML, PNG
ePUB, XML, XPS, JPG...

Aspose.Pdf

PDF, XML, XSL-FO, HTML, BMP
JPG, PNG, ePUB...

Aspose.Slides

PPT, POT, ODP, XPS, HTML
PNG, PDF...

Aspose.BarCode

JPG, PNG, BMP, GIF, TIF, WMF
ICON...

Aspose.Tasks

XML, MPP, SVG, PDF, TIFF
PNG...

Aspose.Email

MSG, EML, PST, MHT
OST, OFT...

Aspose.Imaging

PDF, BMP, JPG, GIF, TIFF
PNG...

+ more!

.NET

Java

Cloud

Download FREE trial at www.aspose.com.

Contact Us:

US: +1 903 306 1676

EU: +44 141 628 8900

AU: +61 2 8006 6987

sales@aspose.com

attributes and fluent syntax to define a mapping between class properties and columns of the underlying table. In the same way, you can define indexes, primary keys, identity columns, default values, and whatever else the underlying database lets you configure in columns and tables. Here's a minimal version of the Region class:

```
public class Region
{
    public Region()
    {
        Languages = "EN";
    }

    [Key]
    public string RegionCode { get; set; }
    public string RegionName { get; set; }
    public string Languages { get; set; }
    ...
}
```

As is, all records in the Regions table will have three nvarchar(MAX) columns (RegionCode, RegionName and Languages) and RegionCode will be set as the primary key column. In addition, any newly created instance of the Region class will have the value "EN" set to the Languages property. This is a way to ensure that "EN" is the default value for the column whenever a new record is added through the code. Note, though, that setting a value in the constructor of a Code-First solution as is done here doesn't automatically add any default value binding in the underlying database configuration.

Naming the Database

In a Code-First solution, all connections to the database pass through the DbContext-derived class, which opens and closes connections appropriately while still publicly exposing the connection as a property for your code to take full control over open and close operations. What about the details of the connection string and, more important, how do you provide details in a parametric way?

When you create a DbContext-derived class, you must provide a constructor. Here's a very common example:

```
public RegionsContext(string conn) : base(conn)
{
    ...
}
```

The DbContext class has a constructor that accepts the connection string as a parameter so the simplest thing to do is that you mirror the functionality of the underlying constructor through the constructor of your derived class. Intelligently, the DbContext class contains some logic to process the string being passed. Any string you pass that's in the form name=XXX is assumed to indicate that the actual connection string can be found in the XXX entry within the connectionstrings section of the application's configuration file (that is, web.config for a Web project). Otherwise, any string passed is assumed to be the name of the database to create. In this case, further details of the connection string, such as credentials and server location, are expected to be in the defaultConnectionFactory block of the entityFramework section in the configuration file. Note that whenever you add

the Entity Framework package to a Visual Studio project, the configuration file is silently modified to support the entityFramework section. **Figure 1** shows the related listing, amended a bit for clarity.

Most of the examples you find about Code First are based on a fixed and constant connection string either referenced from the configuration file or explicitly passed to the context class. The net effect is that Code First creates the database using the provided connection string the first time the application is run. Let's delve a bit deeper into this aspect.

The DbContext class supports four initialization strategies, as listed in **Figure 2**.

According to the default behavior CreateDatabaseIfNotExists, every time the context class is created, it checks whether the referenced database exists and is reachable. If not, it creates it. If the database exists and is reachable but doesn't have a schema compatible with the public layout of entity classes, then it throws an exception. To remove the exception, you have to edit the entity class or, more likely, edit the schema of the database either through the database programming interface or an Entity Framework migration script.

According to the default behavior, CreateDatabaseIfNotExists, every time the context class is created, it checks whether the referenced database exists and is reachable.

I find this option ideal for when the application reaches the production level. During the development stage, though, I prefer the DropCreateDatabaseIfModelChanges option, which essentially shields you from any database maintenance chores: All you do is tweak the entity classes as appropriate and Entity Framework fixes the database the next time you hit F5 in Visual Studio. To activate the initialization strategy of your choice you add the following line to the constructor of the custom DbContext class:

Figure 2 Code-First Database Initialization Strategies

Strategy	Description
CreateDatabaseIfNotExists	Checks if the database exists and creates it if it can't find one. If the database exists but has an incompatible schema then it throws an exception. <i>Note: This is the default initializer.</i>
DropCreateDatabaseIfModelChanges	Creates the database if it doesn't exist. If the database exists but has an incompatible schema then it drops the existing database and creates a new one.
DropCreateDatabaseAlways	Every time you run the application it drops an existing database and recreates it.
Custom initializer	Custom initializer class you write to work out just the behavior you wish to have that none of the other options offer. <i>Note: You must use this option to add some master content to the database.</i>


```
Database.SetInitializer<YourDbContext>(
    new DropCreateDatabaseIfModelChanges<YourDbContext>());
```

Note that you can also set the database initializer in the configuration file, which might be a good idea if you plan to use different strategies in production and development.

At the end of the day, Code First makes writing multi-tenant and multi-customer applications no harder than applications specifically written for a well-known configuration and client.

In summary, Code First enables you to write an application that automatically creates all of its database tables the first time you run it. In other words, all you have to do is copy files and binaries and then launch it. This behavior, however, works at its best if the system is built for a single customer. When you have a multi-customer system, the best you can do is use a setup utility.

One of the reasons for taking a slightly different approach is that you might want to name the database differently; for example, you might want to add a customer-specific prefix to the name. **Figure 3** shows the skeleton of such a command-line utility. The program takes the customer prefix from the command line, formats the database name as appropriate and then triggers the DbContext-derived class, which recreates the database and fills it with appropriate initial data.

Initial Fill of the Database

Any system designed to accommodate the needs of multiple customers in the same business domain must have a number of tables dedicated to storing options and preferences that differ from customer to customer. This information must be provided in some way upon installation of the software. Realistically, part of the initial load of the database is shared by all installations, but part of it is customer-specific. The part that depends on customer's data is commonly imported from external sources and requires an ad hoc routine, whether it's a script of some kind or compiled code. Depending on the context, it might not even be out of place

Figure 3 Customer-Specific Name of the Database

```
class Program
{
    static void Main(string[] args)
    {
        var prefix = args[0];           // boundary checks skipped
        var dbName = String.Format("yourdb_{0}", prefix);

        using (var db = new YourDbContext(dbName))
        {
            // Fill the database in some way
        }
    }
}
```

thinking of some dependency injection mechanism to generalize the structure of importers within the setup utility that initializes the database. As far as static database content is concerned, though, Code First has ad hoc services to offer.

Custom Initializers

To stuff data into the database during the initialization process, it's required that you create a custom database initializer as described in **Figure 2**. A custom initializer is a class that inherits from one of the predefined initializers such as `DropCreateDatabaseIfModelChanges`. The only strict requirement for this class is overriding the `Seed` method:

```
public class YourDbInitializer : DropCreateDatabaseAlways<YourDbContext>
{
    protected override void Seed(YourDbContext context)
    {
        ...
    }
}
```

In the implementation of the `Seed` method, you execute any code that populates the tables of the database using the provided `DbContext` to access it. That's all of it.

If you're planning a multi-customer application, defining a custom initializer is a good move because it gives you a single point to focus on to shape up the initial form of the database on a per-customer basis. The initializer is a plain C# class so it might be empowered using dependency injection tools to connect it to specific pieces of logic that can import data from just where they live.

Last but not least, database initializers can be disabled entirely so that the setup of the database remains a completely distinct operation—perhaps even managed by a different IT or DevOps team. To instruct the Code-First framework to ignore any initializers, you need to follow code in the constructor of the custom `DbContext` class:

```
Database.SetInitializer<YourDbContext>(null);
```

For example, this is a safe option to use when you release updates to existing systems. Disabling initializers ensures you never lose any of the existing data no matter what.

Wrapping Up

At the end of the day, Code First makes writing multi-tenant and multi-customer applications no harder than applications specifically written for a well-known configuration and client. All that's required is a bit of knowledge about assignment of connection strings and the initialization process. In Entity Framework Core, the core principles remain unaltered even though the details of how it ultimately works are different. In particular, the new `DbContext` class features an `OnConfiguring` overridable method through which you connect the context to the database provider of choice and pass it credentials, and whatever else. ■

DINO ESPOSITO is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.

THANKS to the following Microsoft technical expert for reviewing this article:
Andrea Saltarello



CQRS and EF Data Models

Command Query Responsibility Segregation (CQRS) is a pattern that essentially provides guidance around separating the responsibility of reading data and causing a change in a system's state (for example, sending a confirmation message or writing to a database), and designing objects and architecture accordingly. It was initially devised to help with highly transactional systems such as banking. Greg Young evolved CQRS from Bertrand Meyer's command-query separation (CQS) strategy, whose most valuable idea, according to Martin Fowler, "is that it's extremely handy if you can clearly separate methods that change state from those that don't" (bit.ly/2cuoVeX). What CQRS adds is the idea of creating entirely separate models for commands and queries.

CQRS has often been put into buckets incorrectly, as a particular type of architecture, or as part of Domain-Driven Design, or as messaging or eventing. In a 2010 blog post, "CQRS, Task-Based UIs, Event Sourcing, Agh!" (bit.ly/1fZwJ0L), Young explains that CQRS is none of those things, but just a pattern that can help with architecture decisions. CQRS is really about "having two objects where there was previously only one." It's not specific to data models or service boundaries, though it can certainly be applied to those parts of your software. In fact, he states that "the largest possible benefit though is that it recognizes that their (sic) are different architectural properties when dealing with commands and queries."

When defining data models (most often with Entity Framework [EF]), I've become a fan of leveraging this pattern—in particular scenarios. As always, my ideas are meant as guidance, not rules, and just as I've chosen to apply CQRS in a way that helps me achieve my architecture, I hope you'll take them and shape them to suit your own particular needs.

Code download available at msdn.com/magazine/1116magcode.

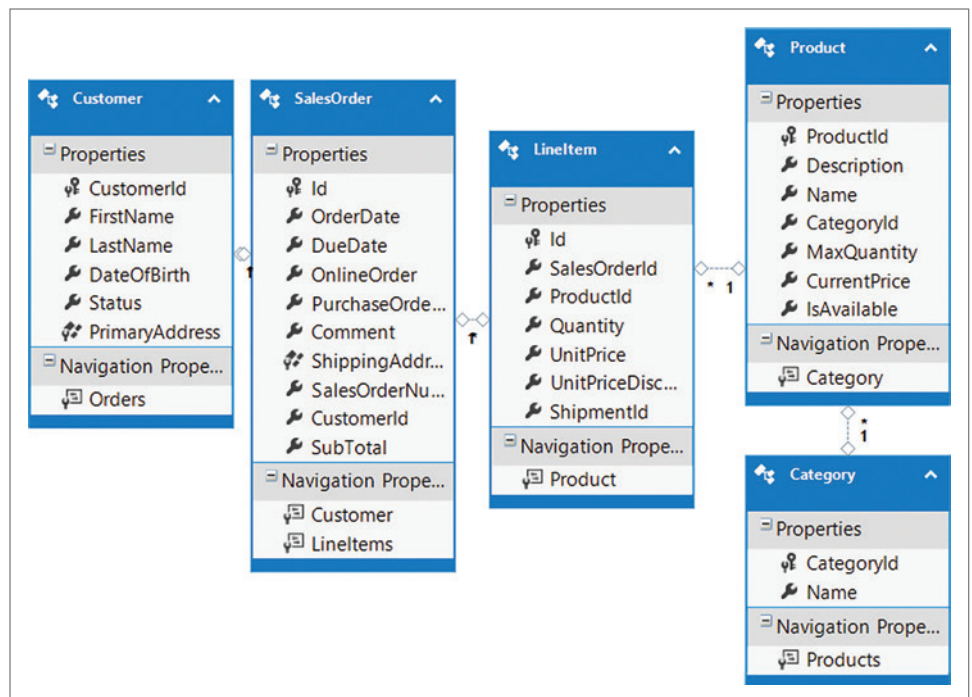


Figure 1 Entity Framework Data Model with Tightly Coupled Relationships

Benefits of Relationship Handling with EF

Entity Framework makes working with relationships at design time so easy. When querying, this is a huge benefit. Relationships that exist between entities allow you to navigate those relationships when expressing queries. Retrieving related data from the database is easy and efficient. You can choose from eager loading with the Include method or with projections, after-the-fact lazy loading or after-the-fact explicit loading. These features haven't changed much since the original version of EF, nor since I wrote about them back in June 2011, "Demystifying Entity Framework Strategies: Loading Related Data" (msdn.com/magazine/hh205756).

The canonical example in the model in **Figure 1** makes querying easy to display the details of a customer's order, the line items and the product names on a page. You can write an efficient query like this:

```
var customersWithOrders = context.Customers
    .Include(c => c.Orders.Select(
        o => o.LineItems.Select(p => p.Product)))
    .ToList();
```

EF will transform this into SQL that will retrieve all of the relevant data in one database command. Then, from the results, EF

We Made WPF GREAT!



Xceed Business Suite for WPF

- 104 high-quality WPF controls and themes
- The fastest and most advanced datagrid
- Widely used controls, over 500K downloads
- Created by WPF pioneers
- 10 years of WPF excellence
- Frequently updated and improved



Customer: Lerman, Julie

Order Date	Customer Status	Customer Discount %	Promo Discount %	Total Discount %	Due Date	Order Source	PO Number	Sales Order
8/15/2016	Silver	0.05	0.1	0.15	8/23/2016	Online		
9/7/2016	Silver	0.05	0	0.05	9/7/2016	InPerson		

Name	Line Total	Quantity	Unit Price	Unit Price Discount %
Coder Keyboard	0	6	55	
Comfy Keyboard	0	7	65	

Figure 2 Data Controls Bound to a Single Object Graph

will materialize the full graphs of customers, their orders, the line items for the orders and even the product details for each line item.

It certainly makes populating a page like the Window Presentation Foundation (WPF) window in **Figure 2** easy. I can do it in a single line of code:

```
customerViewSource.Source = customersWithOrders
```

Here's another benefit that developers love: When creating graphs, EF will work out the back and forth to the database to insert the parent, return the new primary key value, and then apply that as the foreign key value to the children before building and executing their insert commands.

It's all quite magical. But magic has its downsides, and in the case of EF data models, the magic that comes from having tightly

Figure 3 The SalesOrder Type Defined To Be Used for Data Reads

```
namespace Order.Read.Domain {
    public class SalesOrder : Entity {
        protected SalesOrder() {
            LineItems = new List<LineItem>();
        }
        public DateTime OrderDate { get; set; }
        public DateTime? DueDate { get; set; }
        public bool OnlineOrder { get; set; }
        public string PurchaseOrderNumber { get; set; }
        public string Comment { get; set; }
        public int PromotionId { get; set; }
        public Address ShippingAddress { get; set; }
        public CustomerStatus CurrentCustomerStatus { get; set; }
        public double Discount {
            get { return CustomerDiscount + PromoDiscount; }
        }
        public double CustomerDiscount { get; set; }
        public double PromoDiscount { get; set; }
        public string SalesOrderNumber { get; set; }
        public int CustomerId { get; set; }
        public double SubTotal { get; set; }
        public ICollection<LineItem> LineItems { get; set; }

        public decimal CalculateShippingCost() {
            // Items, quantity, price, discounts, total weight of item
            // This is the job of a microservice we can call out to
            throw new NotImplementedException();
        }
    }
}
```

bound relationships can result in side effects when it's time to perform updates, and sometimes even with queries. A notable side effect can happen when you attach reference data to a new record using a navigation property and then call `SaveChanges`. As an example, you might create a new line item and set its `Product` property to an instance of an existing product that came from the database. In a connected app, such as a WPF app, where EF may be tracking every change to its objects, EF will get that the product was pre-existing. But in disconnected scenarios where EF begins tracking the objects only after the changes have been made, EF will assume that the product, like the

line item, is new and will insert it into the database again. There are workarounds for these problems, of course. For this problem, I always recommend setting the foreign key value (`ProductId`) instead of the instance. There are also ways to track the state and sort things out with EF prior to saving the data. In fact, my recent column, "Handling the State of Disconnected Entities in EF" (msdn.com/magazine/mt694083), shows a pattern for doing that.

Here's another common pitfall: navigation properties that are required. Depending on how you're interacting with an object, you may not care about the navigation property—but EF will certainly notice if it's missing. I wrote about this type of problem in another column, "Making Do with Absent Foreign Keys" (msdn.com/magazine/hh708747).

So, yes, there are workarounds. But you can also leverage the CQRS pattern to create cleaner and more explicit APIs that don't require workarounds. This also means that they will be more maintainable and less prone to additional side effects.

Applying CQRS Pattern for DbContext and Domain Classes

I've often used the CQRS pattern to help me get around this problem. Granted that it does mean that whatever models you're breaking up will result in twice as many classes (although not necessarily twice as much code). Not only do I create two separate `DbContext`s, but quite often I'll end up with pairs of domain classes, each focused on the relevant tasks around reading or writing.

I'll use as my example a model that's slightly different from the simpler one I already presented. This example comes from a sizable solution I built for a recent Pluralsight course. In the model, there's a `SalesOrder` class that acts as the aggregate root in the domain. In other words, the `SalesOrder` type controls what happens to any of the other related types in the aggregate—it controls how `LineItems` are created, how discounts are calculated, how a shipping address is derived and so forth. If you think about the tasks I just mentioned, they're focused more on order creation.

You don't really need to worry about the rules around creating a new line item for an order when you're simply reading the order information from the database.

On the other hand, when viewing data, there may be a lot more interesting information to see than I care about when I'm just pushing data into the database.

A Model for Queried Data

Figure 3 shows the `SalesOrder` type in the `Order.Read.Domain` project of my solution. There are a lot of properties here and only a single method for creating better display data. You don't see business rules in here because I don't have to worry about data validation.

Compare this to the `SalesOrder` in **Figure 4**, which I've defined for scenarios where I'll store `SalesOrder` data to the database—whether it's a new order or one that I'm editing. There's a lot more business logic in this version. There's a factory method along with a private and protected constructor that ensure that an order can't be created without particular data being available. There are methods with logic and rules for how a new line item can be created for an order, as well as how to apply a shipping address. There's a method to control how and when a particular set of order details can be modified.

The write version of `SalesOrder` is more complex. But if I ever need to work on the read version, I won't have all of that extraneous write logic in my way. If you're a fan of the guidance that readable code is code that's less prone to errors, you may, like me, have yet another reason to prefer this separation. And surely someone like Young would think even this class has way too much logic in it. But for our purposes, this will do.

The CQRS pattern lets me focus on the problems of populating a `SalesOrder` (which, in this case, are few) and the problems of building a `SalesOrder` separately when defining the classes. These classes do have some things in common. For example, both versions of the `SalesOrder` class define a relationship to the `LineItem` type with an `ICollection<List>` property.

Now let's take a look at their data models; that is, the `DbContext` classes I use for data access.

The `OrderReadContext` defines a single `DbSet`, which is for the `SalesOrder` entity:

```
public DbSet<SalesOrder> Orders { get; set; }
```

EF discovers the related `LineItem` type and builds the model shown in **Figure 5**. However, as EF requires the `DbSet` to be exposed, it also makes it possible for anyone to call `OrderReadContext.SaveChanges`. This is where layers are your friend. Andrea Saltarello provides a

Figure 4 The `SalesOrder` Type for Creating and Updating Data

```
namespace Order.Write.Domain {
    public class SalesOrder : Entity {
        private readonly Customer _customer;
        private readonly List<LineItem> _lineItems;

        public static SalesOrder Create(IEnumerable<CartItem>
            cartItems, Customer customer) {
            var order = new SalesOrder(cartItems, customer);
            return order;
        }

        private SalesOrder(IEnumerable<CartItem> cartItems, Customer customer) : this(){
            Id = Guid.NewGuid();
            _customer = customer;
            CustomerId = customer.CustomerId;
            SetShippingAddress(customer.PrimaryAddress);
            ApplyCustomerStatusDiscount();
            foreach (var item in cartItems)
            {
                CreateLineItem(item.ProductId, (double) item.Price, item.Quantity);
            }
            _customer = customer;
        }

        protected SalesOrder() {
            _lineItems = new List<LineItem>();
            Id = Guid.NewGuid();
            OrderDate = DateTime.Now;
        }

        public DateTime OrderDate { get; private set; }
        public DateTime? DueDate { get; private set; }
        public bool OnlineOrder { get; private set; }
        public string PurchaseOrderNumber { get; private set; }
        public string Comment { get; private set; }
        public int PromotionId { get; private set; }
        public Address ShippingAddress { get; private set; }
        public CustomerStatus CurrentCustomerStatus { get; private set; }

        public double Discount{
            get { return CustomerDiscount + PromoDiscount; }
        }

        public double CustomerDiscount { get; private set; }
        public double PromoDiscount { get; private set; }
        public string SalesOrderNumber { get; private set; }

        public int CustomerId { get; private set; }

        public double SubTotal { get; private set; }

        public ICollection<LineItem> LineItems {
            get { return _lineItems; }
        }

        public void CreateLineItem(int productId, double listPrice, int quantity)
        {
            // NOTE: more rules to be implemented here
            var item = LineItem.Create(Id, productId, quantity, listPrice,
                CustomerDiscount + PromoDiscount);
            _lineItems.Add(item);
        }

        public void SetShippingAddress(Address address) {
            ShippingAddress = Address.Create(address.Street, address.City,
                address.StateProvince, address.PostalCode);
        }

        public bool HasLineItems(){
            return LineItems.Any();
        }

        public decimal CalculateShippingCost() {
            // Items, quantity, price, discounts, total weight of item
            // This is the job of a microservice we can call out to
            throw new NotImplementedException();
        }

        public void ApplyCustomerStatusDiscount() {
            // The guts of this method are in the sample
        }

        public void SetOrderDetails(bool onLineOrder,
            string PONumber, string comment, int promotionId, double promoDiscount){
            OnlineOrder = onLineOrder;
            PurchaseOrderNumber = PONumber;
            Comment = comment;
            PromotionId = promotionId;
            PromoDiscount = promoDiscount;
        }
    }
}
```


great way to encapsulate the DbContext so that only the DbSet is exposed and developers (or future you) using this class don't have direct access to the OrderReadContext. This can help to avoid accidentally calling SaveChanges on the read model.

A simple example of such a class is:

```
public class ReadModel {
    private OrderReadContext readContext = null;
    public ReadModel() {
        readContext = new OrderReadContext();
    }
    public IQueryable<SalesOrder> Orders {
        set {
            return readContext.Orders;
        }
    }
}
```

Another protection you can add to this implementation is to take advantage of the fact that SaveChanges is virtual. You can override SaveChanges so that it never calls the internal DbContext.SaveChanges method.

Leverage the CQRS pattern to create cleaner and more explicit APIs that don't require workarounds.

The OrderWriteContext defines two DbSet: not just one for SalesOrder, but another for the LineItem entity:

```
public DbSet<SalesOrder> Orders { get; set; }
public DbSet<LineItem> LineItems { get; set; }
```

Already that's interesting, as I didn't bother exposing a DbSet for LineItems in the other DbContext. In the OrderReadContext, I'll query only through the SalesOrders. I won't ever query directly against the LineItems, so there's no need to expose a DbSet for that type. Remember in the query to populate the WPF window as in Figure 2. I eager-loaded the LineItems via the Orders DbSet.

The other important logic in the OrderWriteContext is that I've explicitly told EF to ignore the rela-

tionship between SalesOrder and LineItem using the fluent API:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
    modelBuilder.Entity<SalesOrder>().Ignore(s => s.LineItems);
}
```

The resulting model looks like Figure 6.

That means I can't use EF to navigate from SalesOrder to LineItem. It doesn't prevent me from doing that in my business logic; as you've seen, I have lots of code in the SalesOrder class that interacts with LineItems. But I won't be able to write a query that navigates through to LineItems, like context.SalesOrders.Include(s=>s.LineItems). That may raise a moment of panic until I remind you that this is the model for writing data, not for reading it. EF can retrieve related data with no problem using the OrderReadContext.

Pros and Cons of a Relationship-Free DbContext for Writes

So what have I gained by separating the writing responsibilities from the querying responsibilities? It's easy for me to see the downsides. I have more code to maintain. More important, EF won't magically update graphs for me. I'll have to do more work manually to ensure that when I'm inserting, updating or deleting data, the relationships are handled properly. For example, if you have code that adds a new LineItem into a SalesOrder, simply writing myOrder.LineItems.Add(someItem) won't trigger EF to push the orderId into the LineItem when it's time to persist the LineItem into the database. You'll have to explicitly set that orderId value. If you look back at the CreateLineItem method of the SalesOrder in Figure 4, you'll see I've got that covered. In my system, the only way to create a new line item for an order is through that very method, which means I can't write code elsewhere that misses that critical step of applying the orderId. Another question you may ask is: "What if I want to change the orderId of a particular line item?" In my system, that's an action that doesn't make a lot of sense. I can see removing line items from orders. I can see adding line items to orders. But there's no business rule that allows for changing the orderId. However, I can't help thinking of

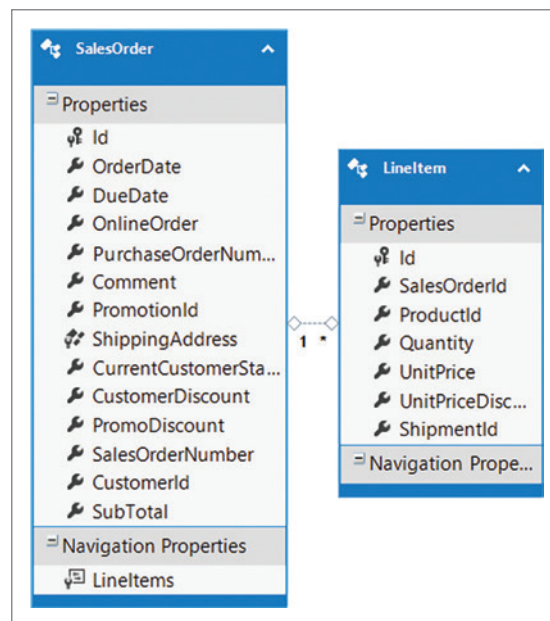


Figure 5 The Data Model Based on the OrderReadContext

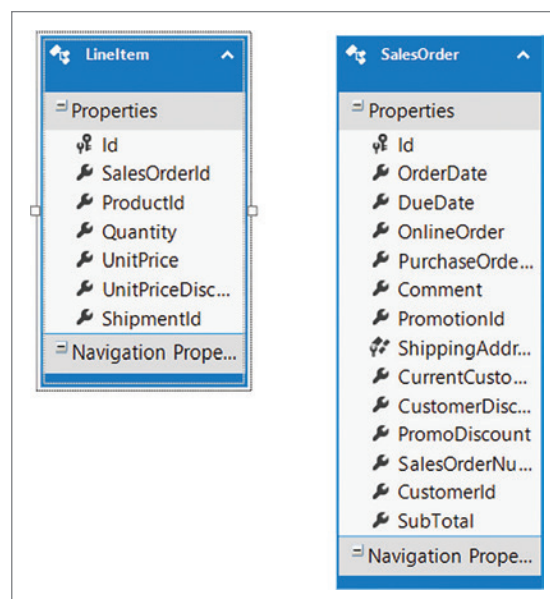


Figure 6 The Data Model Based on the OrderWriteContext

these “what ifs” because I’m so used to just building these capabilities into my data model.

In addition to the explicit control I have over the relationships, breaking up the read and write logic also gets me thinking about all the logic I add to my data models by default, when some of that logic will never be used. And that extraneous logic may be forcing me to write workarounds to avoid its side effects.

In addition to the explicit control
I have over the relationships,
breaking up the read and write
logic also gets me thinking about
all the logic I add to my data
models by default, when some of
that logic will never be used.

The problems I brought up earlier about reference data being re-added to the database accidentally or null values being introduced when you’re reading data you don’t intend to update—these problems will also disappear. A class defined for reading may include values that you want to see but not update. My SalesOrder example doesn’t have this particular problem. But a write class could avoid including properties you may want to view but not update and, therefore, avoid overwriting ignored properties with null values.

Make Sure It’s Worth the Effort

CQRS can add a lot of work to your system development. Be sure to take a look at articles that provide guidance on when CQRS might just be overkill for the problem you’re solving, such as the one by Udi Dahan at bit.ly/2blbd7i. Dino Esposito’s “CQRS for the Common Application” (msdn.magazine/mt147237) also provides insight. My particular use of this pattern isn’t what you might think of as full-blown CQRS, but being given “permission” to split up the reads and writes by CQRS has helped me reduce the complexity of solutions where an overreaching data model had been getting in the way. Finding a balance between writing extra code to get around side effects or writing extra code to provide cleaner, more direct paths to solving the problem takes some experience and confidence. But sometimes your instinct is the best guide. ■

JULIE LERMAN is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of “Programming Entity Framework,” as well as a Code First and a DbContext edition, all from O’Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.

THANKS to the following technical expert for reviewing this article:
Andrea Saltarello (Managed Designs)

msdnmagazine.com



dtSearch®

Instantly Search Terabytes of Text

dtSearch’s document filters
support popular file types, emails
with multilevel attachments,
databases, web data

Highlights hits in all data types;
25+ search options

With APIs for .NET, Java and C++.
SDKs for multiple platforms.
(See site for articles on faceted
search, SQL, MS Azure, etc.)

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS

Seeing the World with Xamarin and the Microsoft Computer Vision APIs

Alessandro Del Sole

In my [last](#) article, I provided a brief introduction to Microsoft Cognitive Services, describing the available RESTful APIs, and showcasing the Face and Emotion APIs in a cross-platform app written with Xamarin.Forms and C# (msdn.microsoft.com/magazine/mt742868). In this article, I'll discuss another important set of APIs, known as Computer Vision. You'll want to read the previous article before you go on with this one because I'm going to assume you're familiar with some concepts about Cognitive Services I explained there, and because I'll reuse some NuGet packages and

code snippets from the previous sample app. With that said, let's start by describing the Microsoft Computer Vision APIs.

Getting Started with the Computer Vision APIs

The Computer Vision APIs allow images to be described and analyzed using natural language. You can upload a picture to the Computer Vision service or point to an image URL, and expect a fully natural description back, without the need to construct and format descriptions on your own. And that's not all. Computer Vision can perform Optical Character Recognition (OCR) over an image that contains text, and it can scan an image to detect faces of celebrities. As with other services, Computer Vision is based on machine learning and supports REST, which means you perform HTTP requests and get back a JSON response. The JSON in **Figure 1** shows an excerpt from the response of Computer Vision analysis over a picture, taken from the official documentation at bit.ly/2a45kQl.

As you can see, the response contains a natural language description of what a celebrity is doing in the picture, with a list of additional information such as tags and picture size and format. And this is just some of the information you can get from the Computer Vision service.

Performing HTTP requests certainly works, but as a .NET developer, you might prefer a different approach. As with the Face and Emotion APIs, Microsoft also offers a portable client library (PCL) for Computer Vision that you can use in your C# applications, including .NET and Xamarin, which lets you invoke the

This article discusses:

- Understanding Microsoft Computer Vision API and how to subscribe to the service
- Creating a cross-platform Xamarin app for picture analysis
- Describing pictures with the Analysis APIs
- Retrieving text from pictures with Optical Character Recognition
- Finding celebrities with a domain-specific model

Technologies discussed:

Xamarin.Forms, Microsoft Cognitive Services, Microsoft Computer Vision API, C#, Visual Studio 2015

Code download available at:

msdn.com/magazine/1116magcode

Figure 1 Result of Computer Vision Analysis of an Image

```
{
  "description": {
    "tags": [
      "person",
      "man",
      "outdoor",
      "window",
    ],
    "captions": [
      {
        "text": "Satya Nadella sitting in front of a building",
        "confidence": 0.3803515599737377
      }
    ]
  },
  "requestId": "ed2de1c6-fb55-4686-b0da-4da6e05d283f",
  "metadata": {
    "width": 1500,
    "height": 1000,
    "format": "Jpeg"
  }
}
```

service through convenient methods using an object-oriented approach. I'll use this library shortly.

Subscribing to the Computer Vision APIs

As with other Cognitive Services, to use the Computer Vision APIs you must register the service and get a secret key to use in your code. To accomplish this, simply go to the Subscriptions page (bit.ly/2b2rKD0) and request a new trial for the Computer Vision APIs. **Figure 2** shows how your subscription appears after registration.

As you can see, you get two secret keys. You'll need one later when it's time to write C# code.

Creating a Xamarin.Forms App and Installing NuGet Packages

Launch Visual Studio 2015 and create a new Xamarin.Forms project using the Blank XAML App (Xamarin.Forms Portable) project template. Call the new project `ComputerVisionSample` and click OK. When ready, install the following NuGet packages:

Microsoft.ProjectOxford.Vision installs the client library for the Computer Vision APIs and must be installed to the PCL project only.

Xam.Plugin.Connectivity contains the Connectivity plug-in for Xamarin.Forms and must be installed to all the projects in the solution. It will be used to check for a network connection before attempting to make requests over the Internet.

Xam.Plugin.Media contains the Media plug-in for Xamarin.Forms and must be installed to all the projects in the solution. It will be used to take and select pictures from shared code, instead of having to write platform-specific code.

Make sure you build the solution at this point, so that all references will be refreshed. Now let's dive into the Computer Vision APIs, analyzing the three key scenarios.

Describing Pictures with the Analysis APIs

The Computer Vision client library exposes a class called `Microsoft.ProjectOxford.Vision.VisionServiceClient`, which is the object you use to send requests to the service, and that exposes properties containing the analysis result. This class will be used in all the scenarios I target. The first scenario is describing pictures, by which I mean obtaining a description of what the picture represents, based on natural, human-readable language. The response the service sends back also contains information, such as dominant colors, faces detected, tags, image type and size, and also whether a picture contains adult or racy content. In order to describe a picture, the `VisionServiceClient` class exposes two methods: `AnalyzeImageAsync` and `DescribeAsync`. The latter provides a smaller set of information, and is typically used to retrieve only a natural language description of the picture, whereas `AnalyzeImageAsync` returns more detailed information. Both methods store their response into an object of type `Microsoft.ProjectOxford.Vision.Contract.AnalysisResult`. I'll be using `AnalyzeImageAsync` in this article. This method has two overloads, one accepting a stream and one accepting a URL pointing to an image; both overloads require specifying the set of information you want to retrieve from the picture. This set of information

is represented by an array of values from the `Microsoft.ProjectOxford.Vision.VisualFeature` enumeration. In the `MainPage.xaml` page of the sample application, I'll implement the picture description. **Figure 3** shows how to implement a detailed analysis method.

Notice how the code uses the Connectivity plug-in to detect a network connection, as I explained in my previous article. There are three key points in **Figure 3**. The first point concerns the information you want to retrieve. The array of `VisualFeature` values contains the most detailed list of information possible, and includes values from the `VisualFeature` enumeration.

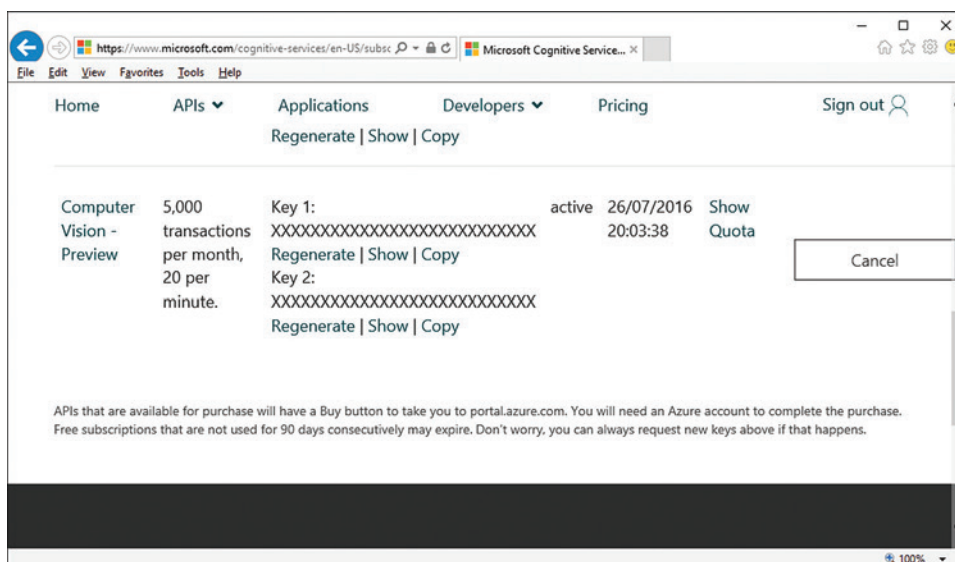


Figure 2 Registering for the Computer Vision APIs

These have self-explanatory names and include detection of adult and racy content, a list of categories for the picture, dominant colors, a natural language description, a list of faces, image information and a list of tags. I've included the Faces value for the sake of completeness, but I won't actually be using this result because you can retrieve more detailed information about faces using the Face API. The second key point is the invocation to `AnalyzeImageAsync`, which sends a stream to the service together with the list of information you want to retrieve, and stores the response into an object of type `AnalysisResult`. This class is the third key point and is defined in **Figure 4**.

To see this definition, right-click the `AnalysisResult` type in the code editor and select `Go To Definition` (or `Peek Definition` if you don't want to leave the active window). As you can see, the definition exposes properties that contain the required information through specialized objects. By using `Go To Definition` on each property type, you can understand how each specialized object is defined and, therefore, how you can use it in your app (with data binding, for example). For instance, the `Description` type is defined as follows:

```
public class Description
{
    public Description();

    public Caption[] Captions { get; set; }
    public string[] Tags { get; set; }
}
```

Figure 3 Analyzing a Picture for Description

```
private async Task<AnalysisResult> AnalyzePictureAsync(Stream inputFile)
{
    // Use the connectivity plug-in to detect
    // if a network connection is available
    // Remember using Plugin.Connectivity directive
    if (!CrossConnectivity.Current.IsConnected)
    {
        await DisplayAlert("Network error",
            "Please check your network connection and retry.", "OK");
        return null;
    }

    VisualFeature[] visualFeatures = new VisualFeature[] { VisualFeature.Adult,
        VisualFeature.Categories, VisualFeature.Color, VisualFeature.Description,
        VisualFeature.Faces, VisualFeature.ImageType, VisualFeature.Tags };

    AnalysisResult analysisResult =
        await visionClient.AnalyzeImageAsync(inputFile,
            visualFeatures);

    return analysisResult;
}
```

Figure 4 The AnalysisResult Class Definition

```
namespace Microsoft.ProjectOxford.Vision.Contract
{
    public class AnalysisResult
    {
        public AnalysisResult();
        public Adult Adult { get; set; }
        public Category[] Categories { get; set; }
        public Color Color { get; set; }
        public Description Description { get; set; }
        public Face[] Faces { get; set; }
        public ImageType ImageType { get; set; }
        public Metadata Metadata { get; set; }
        public Guid RequestId { get; set; }
        public Tag[] Tags { get; set; }
    }
}
```

Here, the most important property is `Captions`, an array of `Caption` objects. Each `Caption` contains a human-readable description the service retrieved from the picture, which is offered through its `Text` property. The `Adult` class is defined as follows:

```
public class Adult
{
    public Adult();

    public double AdultScore { get; set; }
    public bool IsAdultContent { get; set; }
    public bool IsRacyContent { get; set; }
    public double RacyScore { get; set; }
}
```

This is a simpler class and exposes two `bool` properties that return `true` if the picture contains adult or racy content, plus two other properties that represent the confidence for that result. This is particularly useful when you want to restrict content availability. Now take a look at the definition of the `Color` class:

```
public class Color
{
    public Color();

    public string AccentColor { get; set; }
    public string DominantColorBackground { get; set; }
    public string DominantColorForeground { get; set; }
    public string[] DominantColors { get; set; }
    public bool IsBWImg { get; set; }
}
```

Figure 5 The UI Definition for Image Description

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:ComputerVisionSample"
    x:Class="ComputerVisionSample.MainPage">

    <StackLayout Orientation="Vertical">
        <Button x:Name="TakePictureButton" Clicked="TakePictureButton_Clicked"
            Text="Take from camera"/>
        <Button x:Name="UploadPictureButton" Clicked="UploadPictureButton_Clicked"
            Text="Pick a photo"/>
        <ActivityIndicator x:Name="Indicator1" IsVisible="False" IsRunning="False" />
        <Image x:Name="Image1" HeightRequest="240" />

    <ScrollView Padding="10">
        <StackLayout>
            <StackLayout Orientation="Horizontal">
                <Label Text="Adult content: "/>
                <Label Text="{Binding Adult.IsAdultContent}"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal">
                <Label Text="Racy content: "/>
                <Label Text="{Binding Adult.IsRacyContent}"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal">
                <Label Text="Description: "/>
                <Label Text="{Binding Description.Captions[0].Text}"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal">
                <Label Text="Accent color: "/>
                <Label Text="{Binding Color.AccentColor}"/>
            </StackLayout>
            <StackLayout Orientation="Horizontal">
                <Label Text="Tags: "/>
                <ListView ItemsSource="{Binding Tags}">
                    <ListView.ItemTemplate>
                        <DataTemplate>
                            <ViewCell>
                                <Label Text="{Binding Name}"/>
                            </ViewCell>
                        </DataTemplate>
                    </ListView.ItemTemplate>
                </ListView>
            </StackLayout>
        </StackLayout>
    </ScrollView>
</ContentPage>
```


We didn't invent the Internet...

...but our components help you power the apps that bring it to business.



TOOLS • COMPONENTS • ENTERPRISE ADAPTERS

- **E-Business**
AS2, EDI/X12, NAESB, OFTP ...
- **Credit Card Processing**
Authorize.Net, TSYS, FDMS ...
- **Shipping & Tracking**
FedEx, UPS, USPS ...
- **Accounting & Banking**
QuickBooks, OFX ...
- **Internet Business**
Amazon, eBay, PayPal ...
- **Internet Protocols**
FTP, SMTP, IMAP, POP, WebDav ...
- **Secure Connectivity**
SSH, SFTP, SSL, Certificates ...
- **Secure Email**
S/MIME, OpenPGP ...
- **Network Management**
SNMP, MIB, LDAP, Monitoring ...
- **Compression & Encryption**
Zip, Gzip, Jar, AES ...



The Market Leader in Internet Communications, Security, & E-Business Components

Each day, as you click around the Web or use any connected application, chances are that directly or indirectly some bits are flowing through applications that use our components, on a server, on a device, or right on your desktop. It's your code and our code working together to move data, information, and business. We give you the most robust suite of components for adding Internet Communications, Security, and E-Business Connectivity to

any application, on any platform, anywhere, and you do the rest. Since 1994, we have had one goal: to provide the very best connectivity solutions for our professional developer customers. With more than 100,000 developers worldwide using our software and millions of installations in almost every Fortune 500 and Global 2000 company, our business is to connect business, one application at a time.

connectivity
powered by 

To learn more please visit our website →

www.nsoftware.com

This class is used to store a list of colors detected in the picture, such as the accent color, dominant foreground and background colors, and an array of dominant colors. It also exposes a property called `IsBWImg`, of type `bool`, which returns `true` if the picture is black and white. Understanding how these objects are defined and the properties they expose will help when you want to present information in the UI via data binding. I'll leave it to you to explore the definition of the other classes that `AnalysisResult` uses to store the analysis information. As it is, the `AnalysisResult` instance can be data-bound to some UI elements to show information very easily, and I'll get to this shortly. Consider **Figure 5**, which shows the full listing for the XAML required to define the UI for the sample app.

UI defines two buttons, one for selecting a picture from the device and one for taking a picture from the camera, plus an `ActivityIndicator` that shows that an operation is in progress. The selected picture is displayed within an `Image` control. (I used these controls in the previous article, too.) Notice how data binding is

expressed within `Label` controls. For example, you can bind the `Text` property directly to `Adult.IsAdultContent` and `Adult.IsRacyContent` from the `AnalysisResult` instance instead of performing complex bindings. Similarly, you can retrieve the natural language description of a picture by binding directly to the `Text` property of the first `Caption` object in the collection of `Captions` exposed by the `AnalysisResult.Description` property. Of course, this is fine if there's just one caption or if you just want to see the first result. However, `Captions` might contain multiple `Caption` objects and, in that case, you might want to choose a different data binding. The same direct binding is made against the `Color.AccentColor` property. For `Tags`, the UI shows the list of tags with a `ListView` control, with a data template that presents a `Label` for each tag name. In the codebehind, you must first implement two event handlers for the buttons, as shown in **Figure 6**. The code uses the `Media` plug-in I already described in my previous article, so I won't cover it here.

Figure 6 Clicked Event Handlers for the Buttons

```
private async void TakePictureButton_Clicked(object sender, EventArgs e)
{
    await CrossMedia.Current.Initialize();

    if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.
        IsTakePhotoSupported)
    {
        await DisplayAlert("No Camera", "No camera available.", "OK");
        return;
    }

    var file = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions
    {
        SaveToAlbum = true,
        Name = "test.jpg"
    });

    if (file == null)
        return;

    this.Indicator1.IsVisible = true;
    this.Indicator1.IsRunning = true;

    Image1.Source = ImageSource.FromStream(() => file.GetStream());
    var analysisResult = await AnalyzePictureAsync(file.GetStream());
    this.BindingContext = analysisResult;

    this.Indicator1.IsRunning = false;
    this.Indicator1.IsVisible = false;
}

private async void UploadPictureButton_Clicked(object sender, EventArgs e)
{
    if (!CrossMedia.Current.IsPickPhotoSupported)
    {
        await DisplayAlert("No upload", "Picking a photo is not supported.", "OK");
        return;
    }

    var file = await CrossMedia.Current.PickPhotoAsync();
    if (file == null)
        return;

    this.Indicator1.IsVisible = true;
    this.Indicator1.IsRunning = true;
    Image1.Source = ImageSource.FromStream(() => file.GetStream());

    var analysisResult = await AnalyzePictureAsync(file.GetStream());
    this.BindingContext = analysisResult;

    this.Indicator1.IsRunning = false;
    this.Indicator1.IsVisible = false;
}
```

The key point here is the invocation of the `AnalyzePictureAsync` method, whose result (an instance of `AnalysisResult`) is assigned to the page as its data source.

The key point here is the invocation of the `AnalyzePictureAsync` method, whose result (an instance of `AnalysisResult`) is assigned to the page as its data source. This enables data binding of the UI elements seen in **Figure 6**. Then you need to declare and instantiate the `VisionServiceClient` class as follows:

```
private readonly VisionServiceClient visionClient;

public MainPage()
{
    InitializeComponent();
    this.visionClient =
        new VisionServiceClient("YOUR-KEY-GOES-HERE");
}
```

Notice that you need to supply one of the secret keys you got when registering for the Computer Vision APIs. Finally, remember to add the proper permissions in the app's manifests. For instance, the Universal Windows Platform (UWP) project requires the Internet, Webcam and Pictures Library capabilities; and the Android project requires the `INTERNET`, `CAMERA`, `READ_EXTERNAL_STORAGE` and `WRITE_EXTERNAL_STORAGE` permissions. Now you can start the application on your favorite device or emulator. **Figure 7** shows the UWP version running in desktop mode, with an image and the requested information.

Among all of the available information, you'll probably be most impressed by the content of the `Description` property of the `AnalysisResult` class, which provides an auto-generated, human-readable description with no effort.

Retrieving Text from Pictures with OCR

OCR is the electronic conversion of an image of text into editable text. Most scanners ship with OCR software that lets you produce editable documents from images containing text, such as magazine pages. The set of Computer Vision APIs offers an OCR service that can retrieve text from within images, no matter the language of the text. OCR basically results in string objects. To understand how the OCR APIs work, let's add a new XAML page to the PCL project. In Solution Explorer, right-click the ComputerVisionSample (Portable) project, select Add | New Item, and then in the Add New Item dialog, select the Forms Xaml Page item available in the Cross-Platform node. Call the new window OcrRecognitionPage. The VisionServiceClient class exposes a method called RecognizeTextAsync, which performs OCR on an image. This method accepts either a stream or a URL pointing to an image, and you can optionally specify the language. If you don't specify a language, RecognizeTextAsync will attempt to automatically detect the language. It returns an object of type Microsoft.ProjectOxford.Vision.Contract.OcrResults, which is a bit complex and deserves more explanation. For now, consider the following code, which invokes the OCR service over a stream and auto-detects the language:

```
private async Task<OcrResults> AnalyzePictureAsync(Stream inputFile)
{
    if (!CrossConnectivity.Current.IsConnected)
    {
        await DisplayAlert("Network error",
            "Please check your network connection and retry.", "OK");
        return null;
    }

    OcrResults ocrResult = await visionClient.RecognizeTextAsync(inputFile);
    return ocrResult;
}
```

Notice how you always check for network connectivity first. If you want to specify a language, you pass to RecognizeTextAsync an instance of the RecognizeLanguage class, as follows:

```
OcrResults ocrResult =
    await visionClient.RecognizeTextAsync(inputFile,
        new RecognizeLanguage() { ShortCode = "it", LongName = "Italian" }
    );
```

The official sample application for WPF at bit.ly/2ahHum3 shows the full list of supported languages and codes. The OcrResults class is defined as follows:

```
public class OcrResults
{
    public OcrResults();

    public string Language { get; set; }
    public string Orientation { get; set; }
    public Region[] Regions { get; set; }
    public double? TextAngle { get; set; }
}
```

The Language, Orientation and TextAngle properties represent the detected language, the orientation and angle of the recognized text. Regions is an array of Region objects. Each Region represents the areas on the image that contain text, and the Region type has a property called Lines, an array of Line objects, each representing a single line of text in the region. Each Line object has a property called Words, an array of Word objects, each representing a single word in the line. This is a slightly complex hierarchy, but it provides extremely accurate results in that you can work with every single word that the API detects. Don't forget to use Go To Definition to investigate each class's definition. Because of this complexity, some parts of the UI will be generated at run time. For now, in the XAML for the new page,

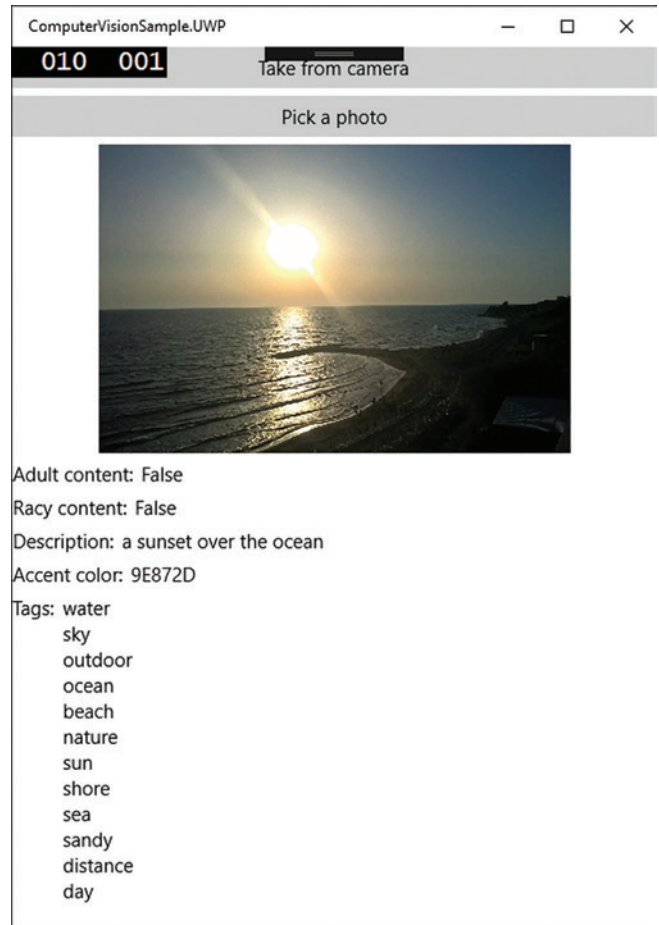


Figure 7 Describing a Picture with the Computer Vision APIs

add the code shown in Figure 8, which declares some familiar controls (two buttons, an Image, an ActivityIndicator) and a StackLayout that will receive the list of lines and words detected. Notice how the code also adds a Label control to display the detected language.

Figure 8 Preparing the UI for Optical Character Recognition

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="ComputerVisionSample.OcrRecognitionPage">

    <StackLayout Orientation="Vertical">
        <Button x:Name="TakePictureButton" Clicked="TakePictureButton_Clicked"
            Text="Take from camera"/>
        <Button x:Name="UploadPictureButton" Clicked="UploadPictureButton_Clicked"
            Text="Pick a photo"/>
        <ActivityIndicator x:Name="Indicator1" IsVisible="False" IsRunning="False" />
        <Image x:Name="Image1" HeightRequest="240" />

        <StackLayout Orientation="Horizontal">
            <Label Text="Language: "/>
            <Label Text="{Binding Language}"/>
        </StackLayout>

        <ScrollView>
            <StackLayout x:Name="DetectedText">

            </StackLayout>
        </ScrollView>

    </StackLayout>
</ContentPage>
```

Figure 9 Iterating Text Regions and Lines

```
private void PopulateUIWithRegions(OcrResults ocrResult)
{
    // Iterate the regions
    foreach (var region in ocrResult.Regions)
    {
        // Iterate lines per region
        foreach (var line in region.Lines)
        {
            // For each line, add a panel
            // to present words horizontally
            var lineStack = new StackLayout
            { Orientation = StackOrientation.Horizontal };

            // Iterate words per line and add the word
            // to the StackLayout
            foreach (var word in line.Words)
            {
                var textLabel = new Label { Text = word.Text };
                lineStack.Children.Add(textLabel);
            }
            // Add the StackLayout to the UI
            this.DetectedText.Children.Add(lineStack);
        }
    }
}
```

As I mentioned, some parts of the UI will be constructed at run time. More specifically, I need to iterate the Regions array, then its nested Lines array, to detect every single word in the Words array. This is demonstrated in **Figure 9**, where a StackLayout is generated for each line.

The remaining code is very simple. First, declare and instantiate the VisionServiceClient class as follows:

```
private readonly VisionServiceClient visionClient;
public OcrRecognitionPage()
{
    InitializeComponent();
    this.visionClient =
        new VisionServiceClient("YOUR-KEY-GOES-HERE");
}
```

You can certainly use the same key you used previously. Next, you can reuse both event handlers shown in **Figure 6**, where you have to replace the following lines:

```
var analysisResult = await AnalyzePictureAsync(
    file.GetStream());
this.BindingContext = analysisResult;
```

with the following new lines:

```
var ocrResult = await AnalyzePictureAsync(
    file.GetStream());
this.BindingContext = ocrResult;
```

```
PopulateUIWithRegions(ocrResult);
```

By doing so, you can bind the OcrResults instance to the UI and then the PopulateUI-WithRegions method will generate new lines with the detected text. For the sake of simplicity, instead of implementing page navigation, you can simply change the startup page in the App.xaml.cs constructor as follows:

```
MainPage = new OcrRecognitionPage();
```

Now start the application again, by choosing your favorite emulator or device. If you select or take a picture, you'll be able to read the text that's printed on it, as shown in **Figure 10**.

In this case, the sample app is running on an Android emulator. Notice how the language has been properly detected as English (en). It's very important to note that the OCR service works well with high-quality images. If the image resolution is poor, the image is blurred, or it contains handwritten or cursive text, the service might return an inaccurate result. It's also worth mentioning that the OCR service can detect words even on images with a multi-color background, not just a solid color. For instance, you can analyze text over the picture of a sunset.

Generally speaking, with a model you can perform specialized analysis over specific categories of images.

Finding Celebrities with a Domain-Specific Model

In the first part of the article, I explained what the Computer Vision APIs offer to describe an image. Description is something that happens at a very high level, and returns general information from an image. Microsoft is also working on offering specialized recognition via the so-called domain-specific models. These allow the return of very specific information from an image, which can be combined with an image description. As of this writing, the only domain-specific model available is celebrity recognition.

By using this model, you can take advantage of the Computer Vision APIs to detect celebrities in a picture. Generally speaking, with a model you can perform specialized analysis over specific categories of images. So my next and final example is recognizing celebrities within pictures. I can't show celebrity pictures for copyright reasons, but you won't have any problem testing the code. Let's start by adding a new XAML page to the PCL project, called CelebrityRecognitionPage. Refer to the previous section for the steps required to add a page. The UI for this page is very simple: It just needs to display the celebrity name in a Label and, of course, it will offer the usual UI elements, as shown in **Figure 11**.

Celebrity recognition is performed using a method from the VisionServiceClient class called AnalyzeImageInDomainAsync. This method requires using the image stream or URL and the domain-specific model for detection. You can retrieve the list of available models by invoking the VisionServiceClient.ListModelsAsync; though, as I mentioned, only the celebrity recognition model is available at the moment. The following code demonstrates how to retrieve the list of models and one specific model:



Figure 10 Performing Optical Character Recognition on an Image



Help & Manual Professional | from \$586.04



Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePub, Kindle or print
- Styles and Templates give you full design control

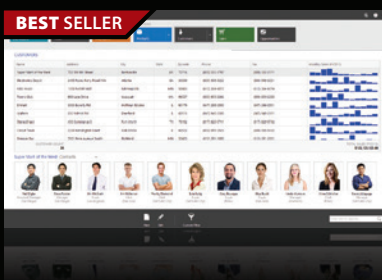


Aspose.Total for .NET | from \$2,939.02



Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

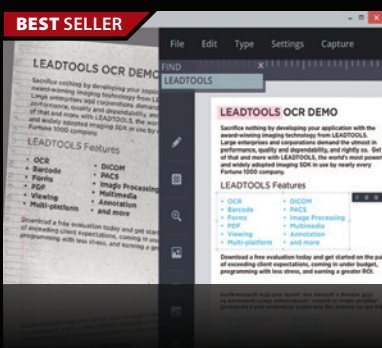


DevExpress DXperience 16.1 | from \$1,439.99



The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics



LEADTOOLS Document Imaging SDKs V19 | from \$2,995.00 SRP



Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

```
private async Task<Model> GetDomainModel()
{
    ModelResult modelResult = await visionClient.ListModelsAsync();
    // At this writing, only celebrity recognition
    // is available. It is the first model in the list
    return modelResult.Models.First();
}
```

The next step is the code that performs recognition, as in the following custom method called `AnalyzePictureAsync`:

```
private async Task<AnalysisInDomainResult> AnalyzePictureAsync(Stream inputFile)
{
    if (!CrossConnectivity.Current.IsConnected)
    {
        await DisplayAlert("Network error",
            "Please check your network connection and retry.", "OK");
        return null;
    }

    AnalysisInDomainResult analysisResult =
        await visionClient.AnalyzeImageInDomainAsync(inputFile, await GetDomainModel());

    return analysisResult;
}
```

The result of image analysis is an object of type `AnalysisInDomainResult` with the following definition:

```
public class AnalysisInDomainResult
{
    public AnalysisInDomainResult();

    public Metadata Metadata { get; set; }
    public Guid RequestId { get; set; }
    public object Result { get; set; }
}
```

The `Result` property contains the actual result of recognition. As you can see, it's of type `System.Object`, which means it contains raw data. More specifically, `Result` stores the JSON response returned by the Computer Vision service. Depending on the number of celebrities detected, this JSON can be very complex, and this is the reason it's an object instead of a more specialized type. It basically defines an array of items, each containing the celebrity name, the face rectangle size, and a value representing the accuracy of the result. For instance, if the result contains one celebrity, the JSON will look similar to the following (where `CelebrityName` stands for the real celebrity name):

```
{
  "celebrities": [
    {
      "name": "CelebrityName",
      "faceRectangle": {
        "left": 169,
        "top": 148,
        "width": 186,
        "height": 186
      },
      "confidence": 0.9064959
    }
  ]
}
```

Figure 11 Preparing the UI for Celebrity Recognition

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="ComputerVisionSample.CelebrityRecognitionPage">
    <StackLayout Orientation="Vertical">
        <Button x:Name="TakePictureButton" Clicked="TakePictureButton_Clicked"
            Text="Take from camera"/>
        <Button x:Name="UploadPictureButton" Clicked="UploadPictureButton_Clicked"
            Text="Pick a photo"/>
        <ActivityIndicator x:Name="Indicator1" IsVisible="False" IsRunning="False" />
        <Image x:Name="Image1" HeightRequest="240" />

        <Label x:Name="CelebrityName"/>
    </StackLayout>
</ContentPage>
```

If the JSON contains multiple celebrities, you can imagine how complex it can be. So an important problem to solve here is retrieving the celebrity name from an `Object` type. This can be done by using the popular `Newtonsoft.Json` library, which is a dependency of the `Microsoft.ProjectOxford.Vision` library and, therefore, is already available in the PCL project. The library provides an object called `JObject`, from the `Newtonsoft.Json.Linq` namespace, which allows parsing JSON markup stored inside a `System.Object` with a method called `Parse`. You can then treat the result as a JSON string and retrieve the desired element with an index. The following method demonstrates how to retrieve a celebrity name from the analysis result:

```
private string ParseCelebrityName(object analysisResult)
{
    JObject parsedJSONResult = JObject.Parse(analysisResult.ToString());

    var celebrities = from celebrity in parsedJSONResult["celebrities"]
        select (string)celebrity["name"];

    return celebrities.FirstOrDefault();
}
```

In this case, I'm assuming a picture contains only one celebrity, so the code invokes `FirstOrDefault` over the result of the LINQ query, but you can work with the query result to see how many celebrities have been detected. The next step is declaring and instantiating the `VisionServiceClient` class, again with the secret key:

```
private readonly VisionServiceClient visionClient;
public CelebrityRecognitionPage()
{
    InitializeComponent();
    this.visionClient = new VisionServiceClient("YOUR-KEY-GOES-HERE");
}
```

At this point, you can add the two-event handler for the buttons `Clicked` event. You can still reuse the code in **Figure 6**, just replacing the following line:

```
this.BindingContext = analysisResult;
```

```
with the following line:
this.CelebrityName.Text = ParseCelebrityName(analysisResult.Result);
```

You can now test the application by selecting a picture of your favorite celebrity and see how the Computer Vision APIs return the exact result.

Wrapping Up

The Computer Vision APIs open up an incredible number of new opportunities, and they provide a really simple way to describe the world, using your app on any platform and on any device. At the Build 2016 conference, Microsoft presented the Seeing AI project, based on several Cognitive Services, including Computer Vision, which is showcased in a short video that gives you a realistic perception of what you can do. Watch it at bit.ly/1qk5ZkJ. ■

ALESSANDRO DEL SOLE has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole is internationally considered a Visual Studio expert, Windows Presentation Foundation and Visual Basic authority, plus he works as a solution developer expert for Brain-Sys (brain-sys.it), focusing on .NET development, training and consulting. You can follow him on Twitter: @progalex.

THANKS to the following Microsoft technical expert for reviewing this article:
James McCaffrey

Enterprise Reporting and Spreadsheets

ar

ActiveReports

The Reporting Platform for All of Your Business Needs

- Sophisticated, fast, and powerful reports
- Visual Studio-integrated report designer
- Comprehensive collection of data visualization controls
- Responsive HTML5 Report Portal for enterprise-wide report delivery
- Royalty-free redistribution

Download your free 30-day trial

activereports.grapecity.com

sp

Spread

Enterprise-Grade .NET Spreadsheet Components

- Offer your users Excel-like spreadsheets with native Microsoft Excel compatibility
- Provide advanced charting and powerful formula library with flexible and familiar spreadsheet architecture
- Create financial modeling and risk analysis, budgeting, insurance, scientific applications and more

Download your free 30-day trial

spread.grapecity.com



© 2016 GrapeCity, inc. All rights reserved. All other product and brand names are trademarks and/or registered trademarks of their respective holders.

Capture and Analyze Brain Waves with Azure IoT Hub

Benjamin Perkins

The brain is the engine that interprets simultaneous input from many sources and many interfaces, and then triggers some kind of action or reaction. Sources like a flower, the sun, or firecrackers and interfaces like smell, sight, and sound can trigger calmness, a physical movement into the shade, or a jerk reaction to a loud sound. As of now, a dependable algorithm like this doesn't yet exist because of the massive number of variables between the source, the human and the interface.

The way to derive this complex algorithm is to get a better understanding of how the brain acts and reacts in numerous situations, like smelling a flower, being in the burning sun, and unexpectedly hearing firecrackers. This article describes how to get more insights on how the brain functions in given scenarios in hopes to someday define an algorithm that reacts dependably in multiple unexpected situations.

This article discusses:

- Capturing, storing and analyzing brain waves
- Configuring the Emotiv SDK
- Creating an Azure IoT Hub and then connecting and uploading data to it

Technologies discussed:

Emotiv SDK, Azure IoT Hub

Code download available at:

msdn.com/magazine/1116magcode

Capturing, Storing and Analyzing Brain Waves

The project described in this article uses numerous technologies to capture, store and analyze brain waves, each of which are briefly described in **Figure 1**. The first two components—capturing the brain waves and storing them into an Azure IoT Hub—are described in the next sections of this article. The remaining three components will be explained in Part 2 of this article in a future issue of *MSDN Magazine*.

The sections are broken down by components, each one including a functional and technical description, plus the details of coding or configuration requirements. The different portions of the solution are ordered in the manner in which I created them; however, it's possible to create them in many different ordered sequences. The technical goal is to upload the brain waves collected using a brain

Figure 1 Components of the Brain Analyzation Project

Components	Role	Brief Description
Emotiv Insights SDK	Capture	A brain interface that converts brain waves to numbers
Azure IoT Hub	Storage	Temporary storage queue for IoT-device-rendered data
SQL Azure	Storage	Highly scalable, affordable and elastic database
Stream Analytics	Storage	An interface between Azure IoT and SQL Azure
Power BI	Analysis	Data analysis tool with simple graphic-based support

Figure 2 Emotiv Insight BCI Readings

Brain Frequencies	State of Mind
ALPHA	Relaxed, high levels of creativity, reflective
LOWBETA	Social activities, excitement, alert
HIGHBETA	Focus, quick thinking, working
GAMMA	Optimal frequency for thinking, active thought
THETA	Sleep, drowsy, meditative and dreaming

computer interface (BCI), store them into a SQL Azure database and analyze the data with Power BI.

Capturing the Brain Waves

When the brain receives input and needs to respond, it decides how to answer by firing electrical currents between neurons called neural oscillations. These neural oscillations are physical movements that cause real, recordable vibrations of different intensity and from different locations in the brain.

An electroencephalograph captures these vibrations and only in the past few years have companies started creating an affordable BCI to capture these brain activities. (There's a list of many of those companies and devices at bit.ly/2c7j4fw.) Additionally, several of these companies have created an SDK for their devices that allow for real-time visualization and storage of brain activity.

I wrote a short post about my initial intent of putting my brain waves into Azure; you can read it at bit.ly/294Hi4R. Notice that I've chosen the Emotiv Insight BCI for this project. This BCI has five electrodes (AF3, AF4, T7, T8, and O1) with each providing state-of-mind readings on five different brain frequencies, as shown in **Figure 2**.

The Emotiv SDK is downloadable from GitHub (github.com/Emotiv) and is easily configurable; this example uses the community-sdk version. While configuring the C# version of the SDK to run with Visual Studio, there were three "gotchas" that were not intuitive:

1. You need to pay close attention to the "bitness" of your Visual Studio project and that the bitness property targets the bitness of the components in No. 3.
2. Make sure the DotNetEmotivSDK.dll is compiled to the same bitness as No. 3.
3. You need to manually copy the edk.dll and the glut32.dll/glut64.dll into the solutions working directory, for example: /bin/Debug or /bin/Release.

To begin, navigate to the C# project in the community-sdk-master\examples\C# folder and open the DotNetEmotivSDK solution in Visual Studio. Set the DotNetEmotivSDK project as the startup project by right-clicking on the project and selecting Set as StartUp Project, then compile the project by pressing Ctrl+Shift+B. Pay special attention to the Platform Target and make sure to keep it consistent during the configuration of the SDK. You should choose either x86 or x64.

Next, create a new console application in Visual Studio and add a reference to the DotNetEmotivSDK.dll that was created during the compilation of the DotNetEmotivSDK project by right-clicking on References and navigating to the ex: \obj\x86\Release directory and selecting the just compiled binary file. Last, copy the edk.dll and the glut*.dll file to the same working directory as the DotNetEmotivSDK.dll was placed. There are numerous copies of the edk.dll and glut*.dll.

Choose the binaries contained in this location of the SDK, community-sdk-master\bin\win32, if you've compiled everything to 32-bit, otherwise, choose the 64-bit version.

Once the SDK is properly configured and your new console application is ready, place using Emotiv in the Program.cs class to reference the capabilities in the library. If desired, view the Brain-ComputerInterface project in the downloadable example code. Pay special attention to GetHeadsetInformation as this is where some pre-validation of the BCI device is executed.

The GetHeadsetInformation method subscribes to the EmoState-UpdatedEventHandler, which is triggered when the ProcessEvents method of the EmoEngine class is called. The GetHeadsetInformation method continues to call ProcessEvents within a while loop until bool stopHeadsetInformation is set to false. When the EmoState-UpdatedEventHandler is triggered, it executes the engine_EmoState-Updated method, which checks the battery level and signal strength. It's important to the validity of the collected BCI data that the battery has an acceptable charge and that there's an adequate Bluetooth 4.0 LE connection between the BCI contacts and the computer.

In the source code, the capturing of the BCI data doesn't begin until those two measurements pass an adequate threshold, for example, chargeLevel > 1 && signalStrength > EdkDll.IEE_SignalStrength_t.BAD_SIG. As long as the signal strength is greater than IEE_SignalStrength_t.NO_SIG, where NO_SIG means there's no signal, the device is considered functional, but not optimal, therefore the signalStrength must equal at least GOOD_SIG before proceeding. Additionally, the maxChargeLevel is five and current charge level greater than one reflects a functional state. The code capturing the brain waves, the battery level, the signal strength and the contact quality for each of the electrodes is shown here:

```
EmoState es = e.emoState;
EdkDll.IEE_SignalStrength_t signalStrength =
    es.GetWirelessSignalStatus(); es.GetBatteryChargeLevel(
        out chargeLevel, out maxChargeLevel);
WriteLine($"AF3: {(int)es.GetContactQuality(
    (int)EdkDll.IEE_InputChannels_t.IEE_CHAN_AF3)}");
WriteLine($"AF4: {(int)es.GetContactQuality(
    (int)EdkDll.IEE_InputChannels_t.IEE_CHAN_AF4)}");
WriteLine($"T7: {(int)es.GetContactQuality(
    (int)EdkDll.IEE_InputChannels_t.IEE_CHAN_T7)}");
WriteLine($"T8: {(int)es.GetContactQuality(
    (int)EdkDll.IEE_InputChannels_t.IEE_CHAN_T8)}");
WriteLine($"Pz: {(int)es.GetContactQuality(
    (int)EdkDll.IEE_InputChannels_t.IEE_CHAN_O1)}");
```

Caution: The BCI can attain readings from the electrodes even though the contact quality is bad. When some of the electrodes are working and capturing data, other electrodes might not be, which is not an ideal situation because the conclusions made later from the data analysis can be wrongly interpreted if all the electrodes

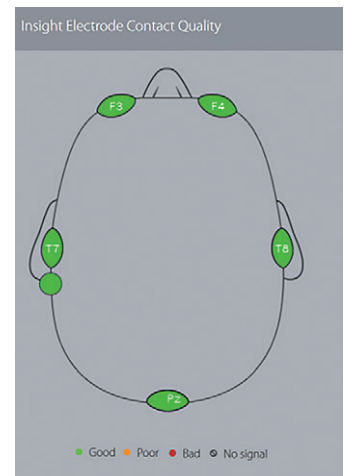


Figure 3 Validate Electrodes on the Brain Interface with BCI

Figure 4 Reading the Frequency Values of Brain Interface Electrodes

```
EmoEngine engine = EmoEngine.Instance;
EdkDll.IEE_DataChannel_t[] channelList = new EdkDll.IEE_DataChannel_t[5]
{
    EdkDll.IEE_DataChannel_t.IED_AF3,
    EdkDll.IEE_DataChannel_t.IED_AF4,
    EdkDll.IEE_DataChannel_t.IED_T7,
    EdkDll.IEE_DataChannel_t.IED_T8,
    EdkDll.IEE_DataChannel_t.IED_O1
};
while (true)
{
    for (int i = 0; i < 5; i++)
    {
        engine.IEE_GetAverageBandPowers(0, channelList[i],
            theta, alpha, low_beta, high_beta, gamma);
        WriteLine($"Channel: {channelList[i]}");
        WriteLine($"Alpha: {alpha[0].ToString()}, Low Beta: {low_beta[0].ToString()}, " +
            $"High Beta: {high_beta[0].ToString()}, Gamma: {gamma[0].ToString()}, " +
            $"Theta: {theta[0].ToString()}");
    }
}
```

are not fully functional during the session. There's no code in the sample to measure and confirm that all electrodes are functional; nevertheless, this should be the case before storing the measurements. An alternative to coding the logic to confirm all electrodes are fully functional prior to running the code is to use the online Emotiv CPANEL, which is accessible from bit.ly/1LZge5T. There, you'll see something similar to **Figure 3**.

Once the engine_EmoStateUpdated method confirms the BCI is functional, it sets stopHeadsetInformation = false, which breaks the while loop in the GetHeadsetInformation method. The C# code to read the frequencies from each of the electrodes is illustrated in **Figure 4** and is found in the GetBrainInterfaceMeasurements

method. The method first creates a single dimensional array of type EdkDll.IEE_DataChannel_t with five reference elements, one reference element per electrode on the device. Finally, program loops through each of the five electrodes and outputs the frequency strengths to the console. Notice that the GetAverageBandPowers method of the EmoEngine class accepts the channel\electrode (channelList[i]) and the frequency variables (theta, alpha, low_beta, high_beta and gamma) into which the numeric representation of the brain wave is to be stored. Each of the readings together with the electrode are rendered to the console window using the static WriteLine method found in the System class.

The console application requires that you have an Emotiv Insight BCI and a valid Bluetooth connection with it. Regardless of the chosen BCI, the principles are the same in these ways:

- Before you begin capturing and storing the data make sure the device is in an optimal and consistent state so that all recorded data is gathered in the same manner.
- Understand how the electrodes are configured and what they measure, then access the measurements and display them, for later storage and analysis.

Once you have the console application working and writing results to the console window, continue to the next section, which discusses how to configure the Azure IoT Hub. How to configure a SQL Azure database into which Stream Analytics inserts the BCI data for analysis and learning will be discussed in Part 2 of this article.

Storing the Brain Waves

In order to store the brain waves collected from the BCI, there are numerous required components. From an individual perspective,

The Parallel Between Coding and the Brain

I don't believe I'm the only person who's made a connection between the construct of code structures and human characteristics. It seems in many ways that the building of code platforms was designed using our own human traits, because the ability to define ourselves in code seems to work so well that it flows, almost without thought. Consider the object-oriented programming term inheritance, where a child class receives a set of attributes and characteristics from a parent class. In the human context, children receive attributes from their parents, like eye color and hair color. Additionally, the ability to walk, blink and smell are examples of methodical characteristics that humans usually possess, as did my parents. However, those characteristics didn't come directly from my parents, they were inherited through many generations, starting from the base Human class itself.

If you were to create a Human class, you'd likely do so by including all the fundamental human attributes and characteristics within that class, like gender, eat, sleep, breathe and so on. Then, you'd create a Parent class inherited from the Human class, with some additional unique or more advanced characteristics such as reflection, speak, love, and so on, assuming, or not, that each generation of the inherited class becomes more sophisticated and complex over time. Inheritance progressively continues into the implementation of a more current Child class.

The way humans speak and communicate changes with each generation, which is where another programming concept called polymorphism arises. Polymorphism means that although the

parent characteristic has the same name, purpose, and intent of the child, it can be performed in a different way and with more inputs so that the outcome is more precise. For example, although the parent has the capacity to speak, the child can have a similar speak method that additionally includes the ability to converse in multiple languages. The additional parameter into the speak method would be language type. This input would not be present in the speak method of the parent. The derived or overloaded speak characteristic could also include some enhanced communication capabilities like facial expression or tone inflection.

Creating these structured classes, the sophisticated methods and the unique set of attributes is a fascinating journey into the learnings of our internal self and existence. Constructing and defining ourselves is the best way to learn what makes us who we are. There is one thing, however, that is quickly realized after the model is built, which is: how to trigger the methods so the Child can actually do something. Instantiating the class is no big deal (Child child = new Child()), but what is the engine that then calls the methods and uses the attributes? Without the engine, all that exists is a motionless and thoughtless entity. While the human engine uses senses like sight, smell and touch to trigger an appropriate method, a computer engine uses data and coded logic to interpret the input as the basis for an action. In order to write that coded logic correctly we would first need a complete understanding of how humans work, which we do not. The missing piece is the brain.



Enterprise-Proven Distributed Caching

Trusted for over a decade, the easiest most powerful in-memory data grid to scale your .NET applications

- ▶ Fast and linearly scalable
- ▶ Enterprise-grade availability
- ▶ Industry-leading ease of use
- ▶ Integrated in-memory computing

Replacing AppFabric Caching?

Try our source-code compatible drop-in.
www.scaleoutsoftware.com/appfabric

Brought to you by the scalability architects

Step up to a battle-tested in-memory data grid that has been hardened by over 425 enterprise customer deployments. ScaleOut's technology makes advanced features such as parallel LINQ query and integrated MapReduce accessible to any .NET developer. Automatic configuration and turnkey global data replication deliver legendary ease-of-use. ScaleOut's world-class support meets the needs of mission-critical applications. Run on premises or in the cloud on Microsoft Azure or Amazon AWS.



ScaleOut Software



Download your free trial today!
www.scaleoutsoftware.com/trial

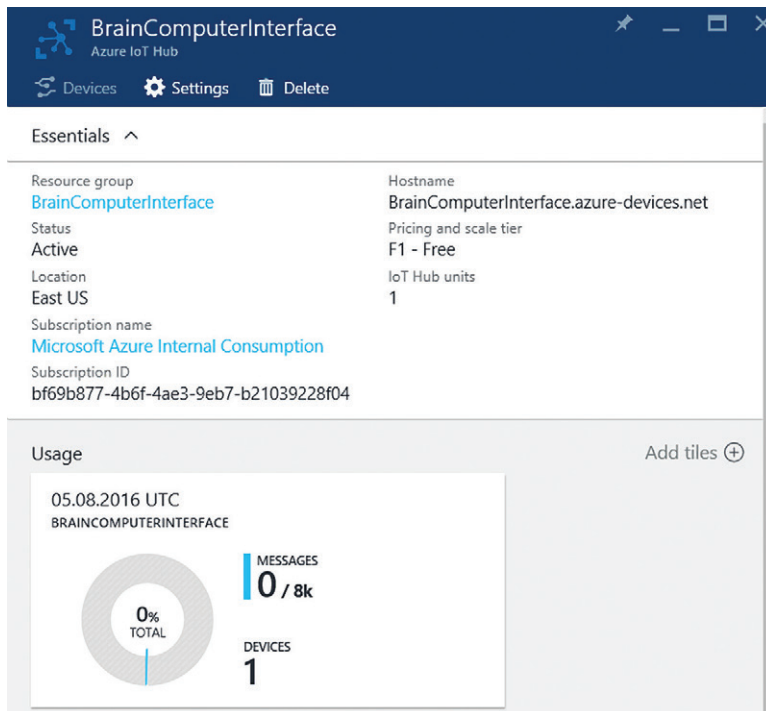


Figure 5 The Details Page of the BCI Azure IoT Hub

you could conceivably make a simple ADO.NET connection to a local SQL database and that's it; however, if you want to let many people with many devices use the application, using an Azure IoT Hub is the best way to go because of its reliability and scalability. The following three components are required to successfully upload and store brain waves:

1. An Azure IoT Hub
 - a. A device identity
 - b. Code to upload the brain wave
2. SQL Azure instance and data table
3. Stream Analytics interface

I'll now discuss in more detail the creation of the Azure IoT Hub.

Create an Azure IoT Hub

An Azure IoT Hub is similar to a message queue in that it temporarily stores multiple rows of data with the expectation that another entity, like a reader or, in this case, a Stream Analytics job, is monitoring the queue and taking an action once the message arrives. The benefit of Azure IoT Hub is that it's extremely resilient and can scale to a very large size in a short amount of time. While testing this solution, I inserted about three rows per second and the client-side record count matched the server-side count exactly. Three events per second is very small; the Azure IoT Hub can handle millions of events per second.

To create an Azure IoT Hub, you need an Azure subscription and access to the Azure Portal at bit.ly/2bA4vAn. Click on the + New menu item and navigate to Internet of Things and select IoT Hub. Enter in the required data and press the Create button. It's possible to have only one Free tier IoT Hub per subscription. The Free tier supports 8,000 events per day. Free tier is the one I picked for this project; however, if you need to insert more events, then choose the appropriate tier. Once created, view the details, as shown in Figure 5.

Once the Azure IoT Hub is created, the next step is to create a unique device identity required for connecting and uploading data to the Azure IoT Hub. The downloadable source contains a console project called BrainComputerInterface-CreateIdentity, which performs this activity. To create your own project, start by creating an empty console application in Visual Studio. Once created, right-click on the project and select Manage NuGet Packages, then search and add the Microsoft.Azure.Devices package with the provided example code; version 1.0.11 is used.

Before starting to code the creation of the device entity, access the Azure IoT Hub and get the connection string by selecting Shared access policies from the Settings blade. Next, select the appropriate Policy explained in the table in Figure 5. Selecting one of the policies shown in the table opens a blade that displays the Permissions and Shared access keys. Copy the Connection string -primary key and use it to set the value of connectionString shown in Figure 7.

To create a device identity, you need a connection string for a policy that has write permission to the identity registry. This means you would use either the iothubowner or registryReadWrite policy. It is highly recommended to

use policies with the least amount of permissions required to perform the desired task; this reduces the chance of unintended actions such as global deletions or updates. Protect the iothubowner connection string parameters and provide it only when the creation of device identities or other administrative activities are required.

View the sample code shown in Figure 7. As this is a simple program, both the _connectionString and Microsoft.Azure.Devices.RegistryManager _registryManager are created as static class variables; it's also fine to create them in the Main method and pass them as method parameters if desired. Instantiate the _registryManager variable by calling CreateFromConnectionStringMethod, then call the Program.AddDeviceAsync method asynchronously.

The Program.AddDeviceAsync method calls the Microsoft.Azure.Devices.RegistryManager.AddDeviceAsync method, passing a new Microsoft.Azure.Devices.RegistryManager.Device. If an identity doesn't already exist, it's created; otherwise, the Microsoft.Azure.Devices.Common.Exceptions.DeviceAlreadyExistsException is thrown. The exception is handled because the code runs within

Figure 6 Connection String Policies, Permissions and Usages

Policy	Permission	Usage
iothubowner	Registry Read/Write, Service/Device Connect	Administration
Service	Service Connect	Sending and receiving on the cloud-side endpoints
Device	Device Connect	Sending and receiving on the device-side endpoints
RegistryRead	Registry Read	Read access to the identity registry
RegistryReadWrite	Registry Read/Write	Read/Write access to the identity registry

a `try{} catch{} code block`. Within the `catch{} block` the `Microsoft.Azure.Devices.RegistryManager.GetDeviceAsync` method is called and, in both cases, whether the `Add` or `Get` methods were called, the device key is rendered to the console.

Once the code is complete and compiles, execute the code and note down the device key, as it's needed to create the `DeviceClient` class that contains the logic to connect and send data to the Azure IoT Hub used in the next section. Also, look again at **Figure 5** and notice that the `Devices` link is initially grayed out. After a device is created, the `Devices` link on the Azure IoT Hub blade is enabled; clicking on it lets you disable/enable the device and retrieve the device key, just in case you missed it in the console window when created.

The code to capture the brain waves has already been written in the previous section. What needs to happen now is that instead of writing the BCI output to the console, write it to the Azure IoT Hub that was just created. In the sample code, there's a project

Figure 7 Create a Device Key for Each Unique Device ID

```
static RegistryManager _registryManager;
static string _connectionString = "the iothubowner connection string";
static void Main(string[] args)
{
    _registryManager = RegistryManager.CreateFromConnectionString(_connectionString);
    AddDeviceAsync().Wait();
    ReadLine();
}
private static async Task AddDeviceAsync()
{
    string deviceId = "ADD UNIQUE DEVICE ID";
    Device device;
    try
    {
        device = await _registryManager.AddDeviceAsync(new Device(deviceId));
    }
    catch (DeviceAlreadyExistsException)
    {
        device = await _registryManager.GetDeviceAsync(deviceId);
    }
    WriteLine($"Generated device key: {device.Authentication.SymmetricKey.PrimaryKey}");
}
```

Figure 8 Inserting the Brain Wave into Azure IoT Hub

```
while (true)
{
    for (int i = 0; i < 5; i++)
    {
        engine.IEE_GetAverageBandPowers(0, channelList[i],
            theta, alpha, low_beta, high_beta, gamma);
        SendBrainMeasurementToAzureAsync(channelList[i].ToString(), theta[0].ToString(),
            alpha[0].ToString(), low_beta[0].ToString(),
            high_beta[0].ToString(), gamma[0].ToString());
    }
}

private static async void SendBrainMeasurementToAzureAsync(string channel,
    string theta, string alpha, string lowbeta, string highbeta,
    string gamma)
{
    // ...
    try
    {
        var brainActivity = new
        {
            ManufacturerId, HardwareId, ActivityId, ChannelId,
            DeviceId, Username, MeasurementDateTime, theta,
            alpha, lowbeta, highbeta, gamma };
        var messageString = JsonConvert.SerializeObject(brainActivity);
        var message = new Message(Encoding.ASCII.GetBytes(messageString));
        await deviceClient.SendEventAsync(message);
    }
    catch (Exception ex)
    {
        //...
    }
}
```

called `BrainComputerInterface` where the `while{} loop` discussed previously in **Figure 2** is changed to call a new method `SendBrainMeasurementToAzureAsync`, as shown in **Figure 8**, which sends the BCI data to the Azure IoT Hub, instead of dumping the brain computer interface reading to the console.

Notice that the `SendBrainMeasurementToAzureAsync` method uses the `Microsoft.Azure.Devices.Client.DeviceClient`, as mentioned earlier, and `Newtonsoft.Json` classes to format the data and add the BCI reading to the cloud. If creating a new project, add these two NuGet packages by right-clicking on the project and selecting `Manage NuGet Packages`.

Now that the code for writing the BCI output to the Azure IoT Hub is complete, you can place the BCI on your head and start the upload. When the `BrainComputerInterface` program starts running, it will ask you to select the scenario in which the brain waves are to be stored. Some examples of those are smelling a flower, being in the sun, hearing a firecracker and so on. Select the scenario, validate that the electrodes/contacts are green (see **Figure 3**) and once the power and sensor modules are ready, the brain waves will start being captured and uploaded to the cloud.

Note that at this point, you would see the Usage meter on the IoT Hub blade change as data is being sent (see **Figure 5**), however, the data would be deleted after about 24 hours as there is, at this point, no database to store the data nor a program to move the messages from the Azure IoT Hub to a permanent storage location. In Part 2, a SQL Azure database is created, followed by the Stream Analytics job, so you can then analyze the data and discover new things.

Wrapping Up

The path this article series should ultimately lead you toward is two-fold. The first is from a cognitive perspective where the more you learn about yourself and how your brain works, the more you can begin to replicate or enhance it to improve your quality of life. Machines are better and faster at completing mathematical computations and they can draw from a much broader knowledge base for decision making, without emotion, than which the human brain is capable. If you can somehow integrate this into your own cognitive being, using some kind of artificial intelligence, then your ability to work faster and more precisely becomes greater.

The other concept is the ability to use thoughts to control items in your day-to-day life. As the proficiency to capture and analyze brain waves increases, the ability to use them with confidence also increases. Once one or more thought processes like push, pull or spin are flawlessly defined, they can be used to control objects or perform activities like changing the television channel or radio channel. It may even be possible to capture a reading and take an action before your own will recognizes that you have a desire to do so. The possibilities are endless. ■

BENJAMIN PERKINS is an escalation engineer at Microsoft and author of four books on C#, IIS, NHibernate and Microsoft Azure. He recently completed coauthoring "Beginning C# 6 Programming with Visual Studio 2015" (Wrox). Reach him at benperk@microsoft.com.

THANKS to the following Microsoft technical expert for reviewing this article:
Sebastian Dau

Orlando
2016

ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

DEC
5-9



Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Journey into Code

Join us as we journey into real-world, practical education and training on the Microsoft Platform. Visual Studio Live! (VSLive!™) returns to warm, sunny Orlando for the conference more developers rely on to expand their .NET skills and the ability to build better applications.



twitter.com/live360
[@live360](https://twitter.com/live360)



facebook.com
Search "Live 360"



linkedin.com
Join the "Live! 360" group!



EVENT PARTNERS



Magenic



SMARTBEAR



IDERA

QuickBase



PLATINUM SPONSORS

GOLD SPONSORS

SUPPORTED BY



TECH EVENTS WITH PERSPECTIVE

6 Great Conferences 1 Great Price

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to five (5) other co-located events at no additional cost:

SQL Server LIVE!
TRAINING FOR DBAS AND IT PROS

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

Office & SharePoint LIVE!
ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

ModernApps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

NEW! APPDEV TRENDS
ENTERPRISE FOCUSED. CODE DRIVEN.

Six (6) events and hundreds of sessions to choose from - mix and match sessions to create your own, custom event line-up - it's like no other conference available today!

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

**REGISTER WITH DISCOUNT
CODE L360NOV AND
SAVE \$300!**



Must use discount code
L360NOV

Scan the QR code to
register or for more
event details.

TURN THE PAGE FOR
MORE EVENT DETAILS



Check Out the Additional Sessions for Devs, IT Pros, & DBAs at Live! 360



Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live!
features 12+ developer
sessions, including:

- Workshop: A Beginner's Guide to Client Side Development in SharePoint - *Mark Rackley*
- Become a Developer Hero by Building Office Add-ins - *Bill Ayres*
- Utilizing jQuery in SharePoint - Get More Done Faster - *Mark Rackley*
- Using the Office UI Fabric - *Paul Schaefflein*
- Enterprise JavaScript Development Patterns - *Rob Windsor*
- Leveraging Angular2 to Build Office Add-ins - *Andrew Connell*
- Webhooks in Office 365 - *Paul Schaefflein*



SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 15+
developer sessions, including:

- What's New in SQL Server 2016 - *Leonard Lobel*
- Powerful T-SQL Improvements that Reduce Query Complexity - *Hugo Kornelius*
- Implementing Data Protection and Security in SQL Server 2016 - *Steve Jones*
- Welcome To The 2016 Query Store! - *Janis Griffin*
- Workshop: Big Data, BI and Analytics on The Microsoft Stack - *Andrew Brust*



TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor features IT Pro
and DBA sessions, including:

- Workshop: 67 VMware vSphere Tricks That'll Pay for This Conference! - *Greg Shields*
- Secure Access Everywhere! Implementing DirectAccess in Windows Server 2016 - *Richard Hicks*
- Getting Started with Nano Server - *Jeffery Hicks*
- Creating Class-Based PowerShell Tools - *Jeffery Hicks*
- Harvesting the Web: Using PowerShell to Scrape Screens, Exploit Web Services, and Save Time - *Mark Minasi*
- PowerShell Unplugged: Stump Don - *Don Jones*
- Facing Increasing Malware Threats and a Growing Trend of BYO with a New Approach of PC Security - *Yung Chou*



ALM / DevOps	Cloud Computing	Mobile Client	Software Practices	Visual Studio / .NET Framework
START TIME	END TIME			
5:00 PM	8:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center		
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk		
START TIME	END TIME			
8:00 AM	5:00 PM	VSM01 Workshop: Distributed Cross-Platform Application Architecture - Rockford Lhotka & Jason Bock		
5:00 PM	6:00 PM	EXPO Preview		
6:00 PM	7:00 PM	Live! 360 Keynote: Digital Transformation - David Foote, Co-founder		
START TIME	END TIME			
8:00 AM	9:00 AM	Visual Studio Live! / Modern Apps Live! - Tim Sneath, Principal		
9:00 AM	9:30 AM	Networking Break • Visit the EXPO		
9:30 AM	10:45 AM	VST01 Building Applications with ASP.NET Core - Scott Allen	VST02 Busy .NET Developer's Guide to Swift - Ted Neward	
11:00 AM	12:15 PM	VST05 Richer MVC Sites with Knockout JS - Miguel Castro	VST06 Busy .NET Developer's Guide to Native iOS - Ted Neward	
12:15 PM	2:00 PM	Lunch • Visit the EXPO		
2:00 PM	3:15 PM	VST09 WCF & Web API: Can We All Just Get Along?!? - Miguel Castro	VST10 Creating Great Looking Android Applications Using Material Design - Kevin Ford	
3:15 PM	4:15 PM	Networking Break • Visit the EXPO		
4:15 PM	5:30 PM	VST13 Busy Developer's Guide to Chrome Development - Ted Neward	VST14 Creating Cordova Apps using Ionic and Angular 2 - Kevin Ford	
5:30 PM	7:30 PM	Exhibitor Reception		
START TIME	END TIME			
8:00 AM	9:15 AM	VSW01 Moving from Angular 1 to Angular 2 - Ben Dewey	VSW02 The Future of Mobile Application Search - James Montemagno	
9:30 AM	10:45 AM	VSW05 Getting Started with Aurelia - Brian Noyes	VSW06 Building Connected and Disconnected Mobile Applications - James Montemagno	
10:45 AM	11:15 AM	Networking Break • Visit the EXPO		
11:15 AM	12:15 PM	Live! 360 Keynote: To Be Announced		
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO		
1:45 PM	3:00 PM	VSW09 Living in a Command Line Web Development World (NPM, Bower, Gulp, and More) - Ben Dewey	VSW10 Understanding the Windows Desktop App Development Landscape - Brian Noyes	
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.		
4:00 PM	5:15 PM	VSW13 Securing Client JavaScript Apps - Brian Noyes	VSW14 Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - Billy Hollis	
8:00 PM	10:00 PM	Live! 360 Dessert Luau - Wantilan Pavilion		
START TIME	END TIME			
8:00 AM	9:15 AM	VSH01 Build Real-Time Websites and Apps with SignalR - Rachel Appel	VSH02 Cognitive Services: Building Smart Applications with Computer Vision - Nick Landry	
9:30 AM	10:45 AM	VSH05 HTTP/2: What You Need to Know - Robert Boedigheimer	VSH06 Building Business Apps on the Universal Windows Platform - Billy Hollis	
11:00 AM	12:15 PM	VSH09 TypeScript and ES2015 JumpStart - John Papa	VSH10 A Developers Introduction to HoloLens - Billy Hollis & Brian Randell	
12:15 PM	1:30 PM	Lunch on the Lanai		
1:30 PM	2:45 PM	VSH13 All Your Tests Are Belong To Us - Rachel Appel	VSH14 Developing Awesome 3D Apps with Unity and C# - Adam Tuliper	
3:00 PM	4:15 PM	VSH17 SASS and CSS for Developers - Robert Boedigheimer	VSH18 From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - Nick Landry	
4:30 PM	5:30 PM	Live! 360 Conference Wrap-Up - Pacifica 6 - Andrew Brust (Moderator),		
START TIME	END TIME			
8:00 AM	5:00 PM	VSF01 Workshop: Angular 2 Bootcamp - John Papa		
12:00 PM	1:00 PM	Lunch		
1:00 PM	5:00 PM	VSF01 Workshop Continues		

Speakers and sessions subject to change

Web Client	Web Server	Windows Client	Modern Apps Live!	Agile	Containerization	Continuous Integration	Java	Mobile	Cloud
------------	------------	----------------	-------------------	-------	------------------	------------------------	------	--------	-------

Pre-Conference: Sunday, December 4, 2016

Pre-Conference Workshops: Monday, December 5, 2016

VSM02 Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - <i>Miguel Castro</i>	VSM03 Workshop: DevOps in a Day - <i>Brian Randell</i>	MAM01 Workshop: Building Modern Mobile Apps - <i>Brent Edwards & Kevin Ford</i>	ADM01 Workshop: Building Teams - <i>Steve Green</i>	ADM02 Workshop: One Codebase to Rule Them All: Xamarin - <i>Fabian Williams</i>
---	---	--	--	--

Is Your IT Career on Track as Businesses Look to Become More Agile?
Chief Analyst and Chief Research Officer, Foote Partners

Day 1: Tuesday, December 6, 2016

Keynote: Faster, Leaner, More Productive: The Next Generation of Visual Studio
Lead Program Manager, Visual Studio Platform, Microsoft

App Dev Trends Keynote: You Are the Future of Enterprise Java!
- *Reza Rahman, Speaker, Author, Consultant*

VST03 What's New in Azure v2 - <i>Eric D. Boyd</i>	VST04 Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - <i>Benjamin Day</i>	MAT01 Modern App Development: Transform How You Build Web and Mobile Software - <i>Rockford Lhotka</i>	ADT01 Hacking Technical Debt - <i>Steve Green</i>	ADT02 Java 8 Lambdas and the Streaming API - <i>Michael Remijan</i>
VST07 Overview of Power Apps - <i>Nick Pinheiro</i>	VST08 Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - <i>Benjamin Day</i>	MAT02 Architecture: The Key to Modern App Success - <i>Brent Edwards</i>	ADT03 Are You A SOLID Coder? - <i>Steve Green</i>	ADT04 PrimeFaces 5: Modern UI Widgets for Java EE - <i>Kito Mann</i>
VST11 Introduction to Next Generation of Azure PaaS - Service Fabric and Containers - <i>Vishwas Lele</i>	VST12 To Be Announced	MAT03 Manage Distributed Teams with Visual Studio Team Services and Git - <i>Brian Randell</i>	ADT05 Agile Architecture - <i>Steve Green</i>	ADT06 Full Stack Java with JSweet, Angular 2, PrimeNG, and JAX-RS - <i>Kito Mann</i>
VST15 Cloud Oriented Programming - <i>Vishwas Lele</i>	VST16 Bringing DevOps to the Database - <i>Steve Jones</i>	MAT04 Focus on the User Experience #FTW - <i>Anthony Handley</i>	ADT07 Crafting Innovation - <i>Steve Green</i>	ADT08 Who's Taking Out the Garbage? How Garbage Collection Works in the VM - <i>Kito Mann</i>

Day 2: Wednesday, December 7, 2016

VSW03 Managing Enterprise and Consumer Identity with Azure Active Directory - <i>Nick Pinheiro</i>	VSW04 Improving Performance in .NET Applications - <i>Jason Bock</i>	MAW01 DevOps, Continuous Integration, the Cloud, and Docker - <i>Dan Nordquist</i>	ADW01 Stop Killing Requirements! - <i>Melissa Green</i>	ADW02 Migrating Customers to Microsoft Azure: Lessons Learned From the Field - <i>Ido Flatow</i>
VSW07 Practical Internet of Things for the Microsoft Developer - <i>Eric D. Boyd</i>	VSW08 I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - <i>Jeremy Clark</i>	MAW02 Mobile Panel - <i>Kevin Ford, Rockford Lhotka, James Montemagno, & Jordan Matthiesen</i>	ADW03 Meeting-Free Software Development in Distributed Teams - <i>Yegor Bugayenko</i>	ADW04 The Essentials of Building Cloud-Based Web Apps with Azure - <i>Ido Flatow</i>
VSW11 How to Scale .NET Apps with Distributed Caching - <i>Iqbal Khan</i>	VSW12 Learn to Love Lambdas (and LINQ, Too) - <i>Jeremy Clark</i>	MAW03 C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - <i>Rockford Lhotka</i>	ADW05 Introduction to Microsoft Office Graph - <i>Fabian Williams</i>	ADW06 Building IoT and Big Data Solutions on Azure - <i>Ido Flatow</i>
VSW15 ARM Yourself for Azure Success - <i>Esteban Garcia</i>	VSW16 Continuous Delivery on Azure: A/B Testing, Canary Releases, and Dark Launching - <i>Marcel de Vries</i>	MAW04 Coding for Quality and Maintainability - <i>Jason Bock</i>	ADW07 As You Think About Azure Databases, Think About DocumentDB - <i>Fabian Williams</i>	ADW08 Where Does JavaScript Belong in the App Store? - <i>Jordan Matthiesen</i>

Day 3: Thursday, December 8, 2016

VSH03 C# Best Practices - <i>Scott Allen</i>	VSH04 Application Insights: Measure Your Way to Success - <i>Esteban Garcia</i>	MAH01 Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - <i>Kevin Ford</i>	ADH01 From VMs to Containers: Introducing Docker Containers for Linux and Windows Server - <i>Ido Flatow</i>	ADH02 Continuous Testing in a DevOps World - <i>Wayne Ariola</i>
VSH07 Debugging Your Way Through .NET with Visual Studio 2015 - <i>Ido Flatow</i>	VSH08 The Ultimate Intro to Docker for Developers - <i>Adam Tuliper</i>	MAH02 Universal Windows Development: UWP for PC, Tablet & Phone - <i>Brent Edwards</i>	ADH03 CQRS 2.0 - Commands, Actors, and Events...Oh My! - <i>David Hoerster</i>	ADH04 Microservices as Chat Bots Are the Future - <i>Yegor Bugayenko</i>
VSH11 Exploring Microservices in a Microsoft Landscape - <i>Marcel de Vries</i>	VSH12 Automated UI Testing for iOS and Android Mobile Apps - <i>James Montemagno</i>	MAH03 Modern Web Development: Building Server Side using .NET Core, MVC, Web API, and Azure - <i>Allen Conway</i>	ADH05 The Curious Case for the Immutable Object - <i>David Hoerster</i>	ADH06 Continuous Integration May Have Negative Effects - <i>Yegor Bugayenko</i>
VSH15 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - <i>Jeremy Clark</i>	VSH16 Writing Maintainable, X-Browser Automated Tests - <i>Marcel de Vries</i>	MAH04 Modern Web Development: Building Client Side using TypeScript and Angular2 - <i>Allen Conway</i>	ADH07 To Be Announced	ADH08 Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - <i>Roy Cornelissen</i>
VSH19 User Experience Case Studies - Good and Bad - <i>Billy Hollis</i>	VSH20 Debugging the Web with Fiddler - <i>Ido Flatow</i>	MAH05 Using All That Data: Power BI to the Rescue - <i>Scott Diehl</i>	ADH09 Get Started with Microsoft PowerApps - <i>Fabian Williams</i>	ADH10 Overcoming the Challenges of Mobile Development in the Enterprise - <i>Roy Cornelissen</i>

Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & John K. Waters

Post-Conference Workshops: Friday, December 9, 2016

VSF02 Workshop: Building Modern Web Apps with Azure - <i>Eric D. Boyd & Brian Randell</i>	MAF01 Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - <i>Jason Bock, Allen Conway, Brent Edwards & Kevin Ford</i>	ADF01 Workshop: Applied Agile - <i>Philip Japikse</i>
VSF02 Workshop Continues	MAF01 Workshop Continues	ADF01 Workshop Continues

Solving Business Problems with the Microsoft Bot Framework

Srikantan Sankaran

People today do everything online or on their phones—buying, selling, banking, researching—and to stay competitive, enterprises constantly need to evolve their applications to deliver the best possible experience to customers who use their services. This involves providing various self-service capabilities, with the convenience of anytime and anywhere access to their data, often from social channels, using voice and messaging. That's a challenge, due to the variety of applications that need to be factored in, and because most of these applications were never designed to handle the scenarios that are faced today. Efforts to address these needs likely require multiple parallel development projects involving considerable resources. The Microsoft Bot Framework, however, can ease the pain.

This article discusses:

- A solution that takes advantage of Microsoft Flow, Azure Search and the Microsoft Bot Framework to solve business problems
- Authoring business process flows using Microsoft Flow to consolidate data from different line-of-business applications
- Configuring Azure Search to index the consolidated data so it can be queried from a bot

Technologies discussed:

Microsoft Bot Framework, Microsoft Flow, Azure Blob Storage, Azure Search, Azure SQL Database

Code download available at:

bit.ly/2cOfANh

The Microsoft Bot Framework provides a platform for organizations to build applications—bots—that consumers can interact with, easily, conversationally, over voice or text, whenever convenient. Without any additional development effort, these bots can be seamlessly accessed from multiple social channels, such as Skype, Slack, Facebook Messenger and so forth. They can give users access to all of their data consolidated from disparate line-of-business (LOB) applications, using technologies like Azure Logic Apps or Microsoft Flow. These technologies ship with connectors for all key business applications in the market today. Azure Search Service provides the powerful Lucene engine that can be used to search the data, both structured and unstructured, in a flexible way. Customers can interact with the bots through natural language conversations, and the Azure Language Understanding Intelligent Service (LUIS) can interpret these conversations for downstream applications to respond to.

In this article and the next, I'll discuss a scenario that illustrates the challenges organizations face today, and how they can create a solution by taking advantage of the Microsoft Bot Framework. You'll find the following references useful as you implement the solution covered in this scenario:

- Indexing Documents in Azure Blob Storage with Azure Search (bit.ly/2d4yr8s)
- Enable and Disable Change Tracking (SQL Server) (bit.ly/2d226wt)

Creating a Solution

The business scenario that forms the background to this article involves an insurance agency that offers different types of insurance

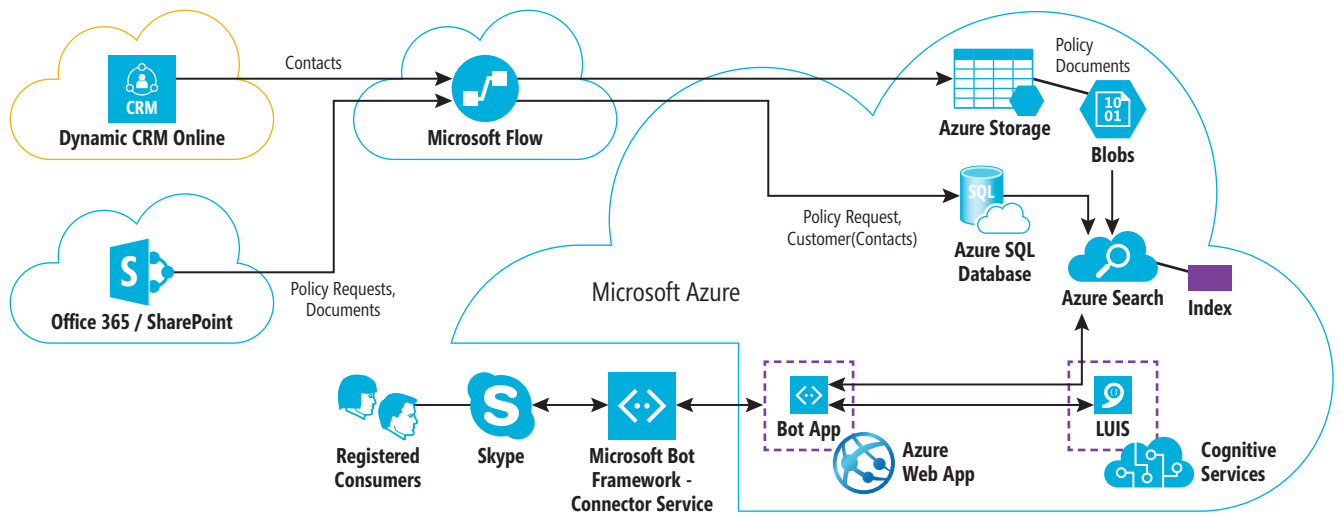


Figure 1 Architecture of the Solution

policies covering vehicles, home, travel and health to consumers. Consumers can register themselves with a Web application using their Microsoft credentials, and submit their request for an insurance policy. A business process workflow takes care of registering the user in Dynamics CRM Online, where profile information is captured, and then stores the insurance policy application in Office 365 SharePoint. An internal workflow process within the organization changes the policy request status over time, finally resulting in its approval and generating a policy document that's stored in Office 365. These steps are outside the scope of this article, which instead focuses on the downstream integration scenarios that kick in when the information is captured in the system, and on sharing the subsequent updates and status with the consumer. **Figure 1** depicts the architecture of the solution for this scenario.

Creating a contact profile for the consumer in Dynamics CRM Online, creating or updating a policy request in Office 365, or uploading policy documents in Office 365 triggers events in business process flows implemented using Microsoft Flow. Through these processes, structured data is synchronized to Azure SQL Database tables, and unstructured data, such as policy documents, is replicated to Azure Blob Storage. Azure Search crawls the data at regular intervals and keeps it current for querying. A bot application is deployed as an Azure Web app, published through the Microsoft Bot Framework Connector service. Through the Connector service, the bot application connects to a Skype

channel, which consumers access to retrieve the status of their insurance policy requests, download the issued policy documents, schedule site inspection visits and so on. Consumers log in with the Microsoft account they registered with when applying for an insurance policy. They can use either messaging on the Skype client or voice commands to interact with the bot. The bot application integrates with the LUIS service, one of the several services available in Microsoft Cognitive Services, to interpret the conversations from the consumer and execute queries on Azure Search.

With this context, here's what has to be done to create the solution:

1. Author business process flows using Microsoft Flow to synchronize and consolidate data from different LOB applications.
2. Configure Azure Search to index the consolidated data so it can be queried from the bot.
3. Configure the LUIS service, and create and train the model for interpreting user conversations from the bot.
4. Build a bot application using the Microsoft Bot Framework and use it from Skype.

I'll describe the first two steps in this article and the remaining two next time.

Insurance Policy Request Synchronization Flow

In this example, policy requests are stored in a custom list in Office 365. A Microsoft Flow process bound to this list is triggered when a policy request is inserted or updated during the approval process, invoking a stored procedure in Azure SQL Database. Separate Microsoft Flow processes are authored for insert-and-update scenarios.

Figure 2 shows how Microsoft Flow Designer lets you select a trigger event on the SharePoint list in Office 365 when creating or updating an item or file.

Figure 3 shows the flow that's triggered when a policy request is inserted into this list. The Microsoft Flow Designer reads the metadata of the stored procedure and automatically displays a form based on its input parameters. Placing the cursor on the form input fields launches the card shown on the right in **Figure 3**, displaying attributes from the preceding activities you can choose from to map to the stored procedure input parameters.

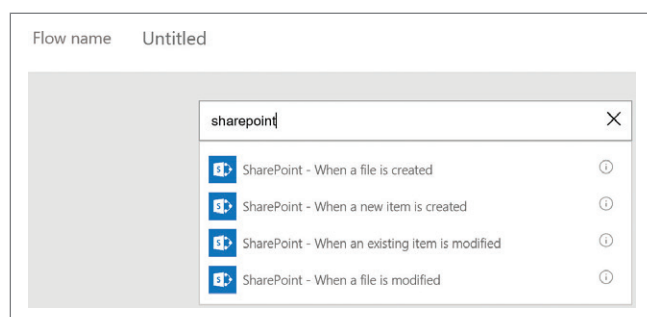


Figure 2 Microsoft Flow Trigger Events

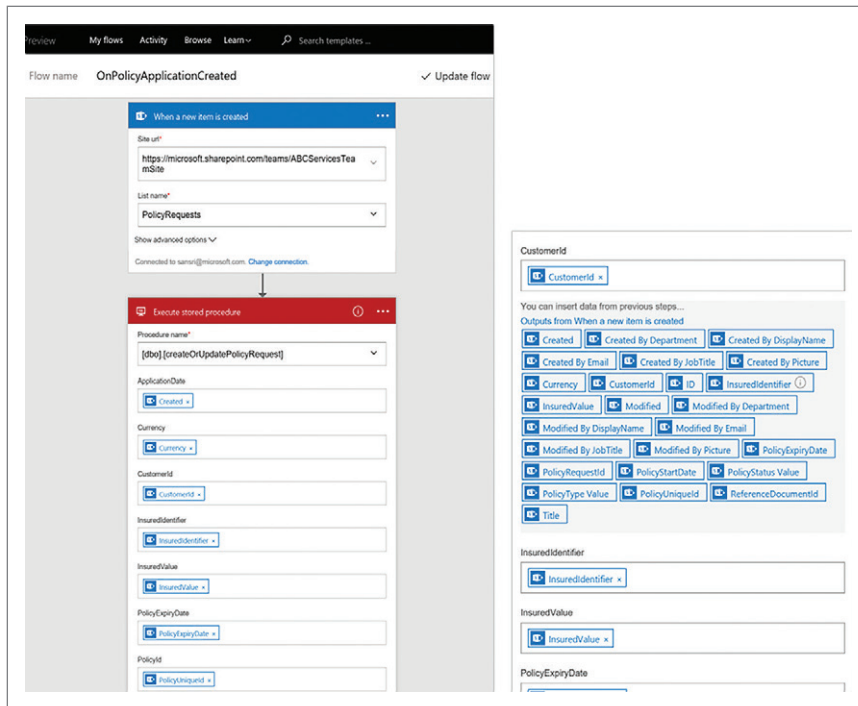


Figure 3 Insurance Policy Request Sync Flow

I'll create an additional flow for syncing updates to ensure the status of the policy approval request is regularly updated from Office 365 to the target Azure SQL database.

Note that each of the connectors used in this scenario requires a connection that must be configured first, and a user account with sufficient rights to read data from a custom list in the Office 365 SharePoint Site. Likewise, the Azure SQL Database connector

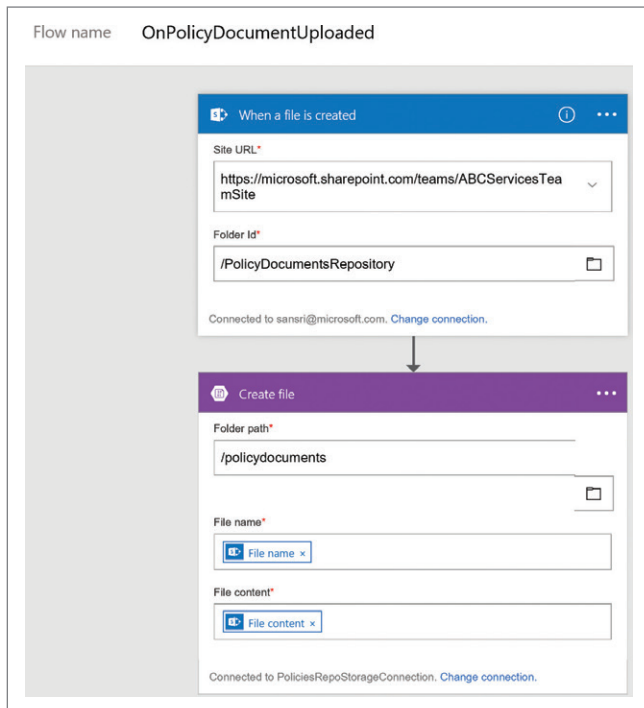


Figure 4 Policy Documents Synchronization Flow

requires user credentials for authentication to the database.

I can test the flow I authored by inserting a record into the custom list. The data should be replicated into the Azure SQL Database.

Policy Documents Synchronization Flow

Once the insurance policy request is approved (on completion of a back-office process that's out of scope of this article), a policy document is generated and uploaded to a document library in Office 365 SharePoint. A Microsoft Flow then replicates the policy document to Azure Blob Storage. Separate Microsoft Flow processes are authored to insert a new document in Azure Storage and to replace the original document if it's updated in Office 365. The document contains certain key words, like the registration number of the vehicle being insured, for example. Later in the article, I'll use Azure Search to execute a full text search within the documents based on specific key words. **Figure 4**

shows the Microsoft Flow implemented to synchronize documents that are uploaded for the first time in Office 365.

I can test the flow by uploading any document to the SharePoint Document Library. The documents should get replicated to the Azure storage container that was configured in the flow.

Customer Profile Data Synchronization

When a customer first registers with the insurance provider, a contact is created in Dynamics CRM Online. A Microsoft Flow is then triggered that uses the Dynamics CRM Online Connector

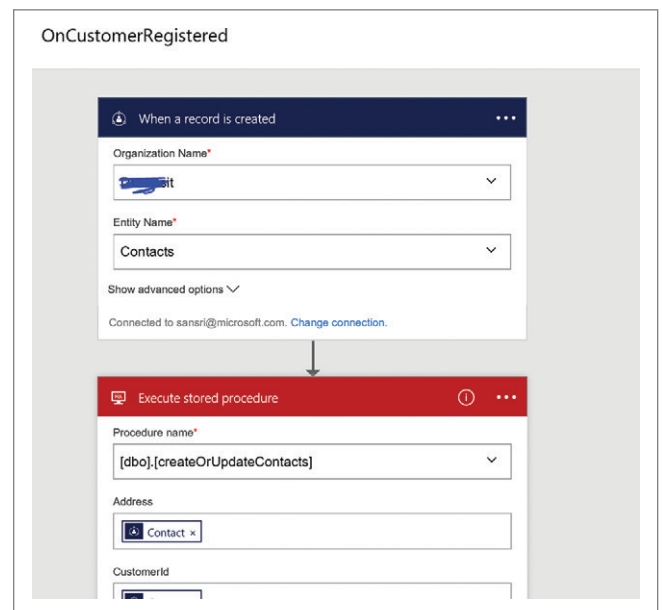


Figure 5 Contact Data Flow

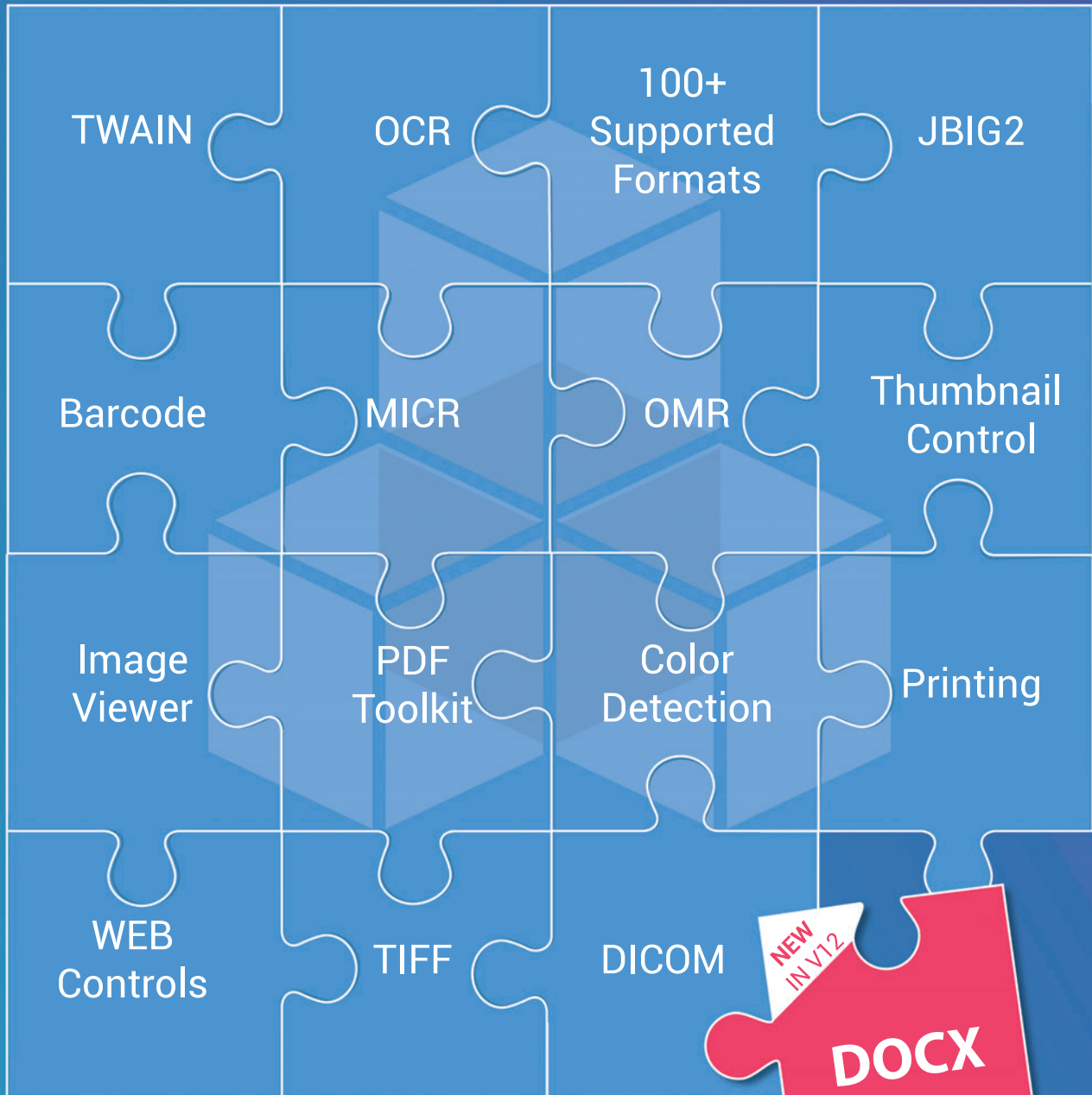
GdPicture.NET



12

100% ROYALTY FREE

Imaging SDK For **WinForms**, **WPF** And **Web** Development



Try **GdPicture.NET V12** for Free for 30 days

www.gdpicture.com

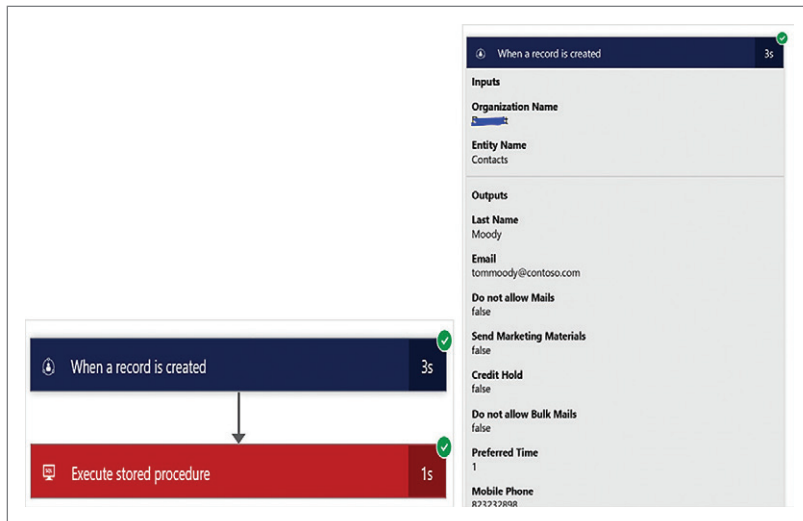


Figure 6 Flow Execution

to pick up the contact and insert it into an Azure SQL Database. **Figure 5** shows how this process can be implemented.

Note that you need to configure a connection to Dynamics CRM Online first, with a user credential that has permission to connect to the organization and a security role that allows access to the data. Further, you should enable change tracking on the table in Dynamics CRM Online where the “insert events” are to be captured, which, in this case, is the Contacts table.

Microsoft Flow lets you view the results of a process execution, with information about the request and response pertaining to each activity in it.

Figure 6 shows the outcome of a contact data synchronization process. Selecting an activity on the left opens a card with the details of the input and output payload, as shown on the right.

To test this flow, I can create a contact from the CRM Online portal. The contact data should get replicated to the Azure SQL Database.

Later, I’ll use the contact data to identify a user logged in to a Skype bot and retrieve the approval status of the user’s insurance policy application request. I’ll also use the contact information to schedule a site inspection visit from the bot.

Preparing the Data Sources for Search

I’ve implemented the process to consolidate the data from all the different LOB applications, but I need to complete certain steps before I consume the data in Azure Search.

Enabling Change Tracking in Azure SQL Database: Azure Search uses the inbuilt indexer for Azure SQL Database to crawl the data and build the index. Change

tracking needs to be enabled on the database and on all the tables, so that Azure Search doesn’t carry out a regenerative index build on them every time:

```
ALTER DATABASE PolicyInfoDB SET CHANGE_TRACKING = ON
(CHANGE_RETENTION = 2 DAYS, AUTO_CLEANUP = ON);
ALTER TABLE CrmCustomerData ENABLE CHANGE_TRACKING
WITH (TRACK_COLUMNS_UPDATED = OFF);
ALTER TABLE PolicyRequests ENABLE CHANGE_TRACKING
WITH (TRACK_COLUMNS_UPDATED = OFF);
```

You can also track deletion of records in database tables and of documents in Azure Storage during the indexing process, through configuration.

Configuring Azure Search to Index the PolicyRequests and CrmCustomerData Tables: Use the Azure Portal to create an entity for your search, then select the option to Import Data from an Azure SQL Database. The wizard takes you through the process of connecting to the database, creating a data source for the search, sampling the data to read the metadata and selecting the necessary

attributes to use, and ending with the index refresh schedule. You configure separate indexes for each of the tables. **Figure 7** shows the search index configuration page for the database.

The Azure Search configuration page detects that change tracking is enabled on the database table being indexed. I’ve set the indexing frequency to Hourly; other options include Once, Daily and Custom.

Once the indexes are configured, Azure Search immediately triggers an indexing operation on the data source. You can check the status of each index to view the number of records that have been populated into the index.

The Azure Search settings blade provides a search explorer that gives you an easy way to set search parameters, view the results and verify the configuration.

Configuring the Azure Search Index for Policy Documents: Configuring Azure Search to search and index Azure Storage blobs

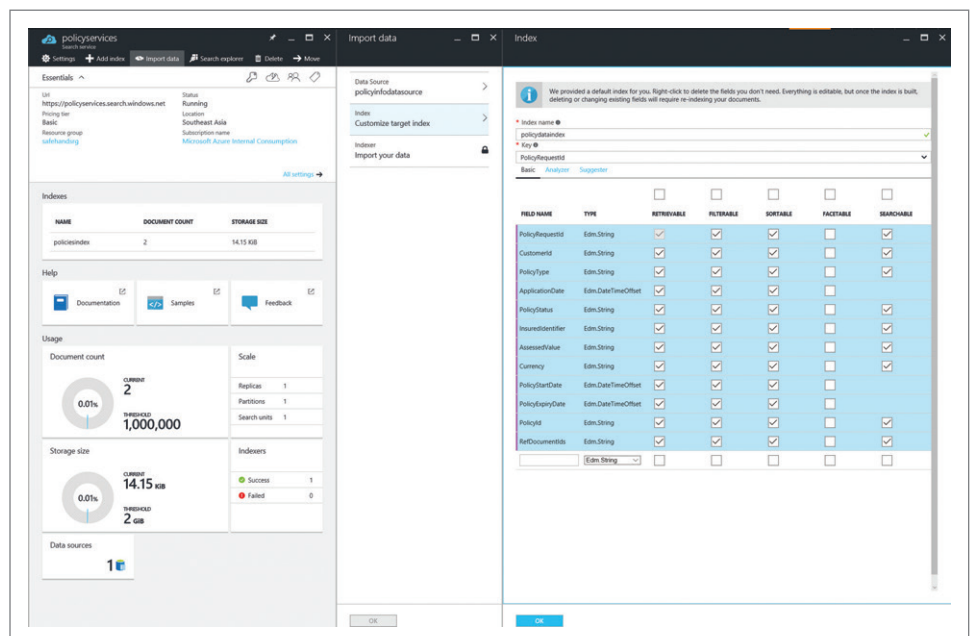


Figure 7 Configure Azure Search to Index Database Table



ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**DEC
5-9**

***REGISTER WITH DISCOUNT
CODE L360NOV AND
SAVE \$300!**

Modern Apps **LIVE!**

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Navigate End-to-End Modern Apps

Presented in partnership with Magenic, Modern Apps Live! brings Development Managers, Software Architects and Development Leads together to break down the complex landscape of mobile, cross-platform, and cloud development and learn how to architect, design and build a complete Modern Application from start to finish.

In-depth and educational sessions taught by the industry's top thought leaders will lay out how to get an app done successfully and at a low cost!



Must use discount code L360NOV
Scan the QR code to register or for
more event details.



A Part of Live! 360: The Ultimate Education Destination

6 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio **LIVE!**

ModernApps **LIVE!**

TECHMENTOR

Office &
SharePoint **LIVE!**

SQL Server **LIVE!**

**APPDEV
TRENDS** NEW!

MODERNAPPSLIVE.COM

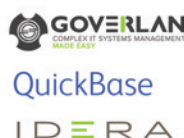
EVENT PARTNERS



PLATINUM SPONSORS



GOLD SPONSORS



SILVER
SPONSOR



SUPPORTED BY



PRODUCED BY



is much like the database configuration just described, though I should note that this feature is still in Preview at this time.

Later in this article I'm going to use Azure Search to query for documents based on the policy document generation date and on key words like the vehicle registration number, which is stored within the documents. Azure Search supports the Apache Lucene Query Language, which provides advanced querying capabilities.

Note that blob storage has a property, `metadata_storage_last_modified`, which is a date-time field. The data type of its counterpart in Azure Search, which the indexer matches it to, is `Edm.DateTimeOffset`, which isn't searchable. Because I want to be able to retrieve policy documents based on the year they're issued, this needs to be a queryable field.

To address this, I could change the data type in the index to `Edm.String`, which is searchable. However, because the indexer uses this field as a flag to determine whether a file has changed since the last indexing operation and whether it should be indexed again, I'd risk breaking that functionality by changing the data type.

Instead, I can add a supplemental field to the index to store the same value using an `Edm.String` data type. However, this means I can't use the Azure Portal UX to configure the index, because it doesn't allow customizing the logic for mapping fields from Azure Storage to the index.

To get around this, I'll use the Postman client (getpostman.com/apps), a free downloadable tool, to invoke the Azure Search Service REST APIs, and to complete the following tasks:

Software Prerequisites for Implementing This Scenario

Apart from an Azure subscription that's required to implement the scenario described here, you'll also need the following development tools and additional software:

- SQL Management Studio for SQL Server 2014 or later to create and connect to the Azure SQL Database.
- A Dynamics CRM Online subscription, with a user account that can connect to the organization where the contacts will be created and has the necessary security roles to read the contacts table. Change tracking should be enabled on this table for its "on Create" trigger to function.
- Access to an Office 365 SharePoint site and a user account with contribute rights to that site.
- A work account to sign in and author the business processes for data synchronization using Microsoft Flow. Get started with it at flow.microsoft.com.
- To use the LUIS service, sign up with your work account (or Microsoft account) at luis.ai. In Part 2 of this article, I'll use this service to model the conversations mentioned in this article.

Note: Microsoft Flow is in preview as of this writing, and can't at this point handle exceptions, implement retry logic or manipulate the code behind the process flow. All the processes presented in this scenario can be authored using Azure Logic Apps, as well, which, aside from providing all the features of Microsoft Flow, supports additional features such as exception handling, using the code editor to manipulate the flow and more.

- **Creating the data source:** This can be done either from the Azure Portal or using the REST API from the Postman client.
- **Configuring the index:** Because I'm defining the fields in the index manually, I can provide names that are more user-friendly than the default Search configuration ones. The "filepath" field will be set as the key field in the index. Notice that there are two placeholder fields for the last modified date, one based on `Edm.DateTimeOffset` and the other on `Edm.String`:

```
{
  "name": "policydocuments-index",
  "fields": [
    { "name": "filepath", "type": "Edm.String", "key": true,
      "searchable": true },
    { "name": "content", "type": "Edm.String", "searchable": true },
    { "name": "filesize", "type": "Edm.String", "searchable": true },
    { "name": "author", "type": "Edm.String", "searchable": true },
    { "name": "filename", "type": "Edm.String", "searchable": true },
    { "name": "lastmoddate", "type": "Edm.String", "searchable": true },
    { "name": "contenttype", "type": "Edm.String", "searchable": true },
    { "name": "modifieddate", "type": "Edm.DateTimeOffset",
      "searchable": false }
  ]
}
```

- **Creating the indexer and mapping fields:** This step creates the indexer, defines the index schedule and maps the fields in Azure Storage to those in the index defined in the previous step:

```
{
  "name": "policydocuments-indexer",
  "dataSourceName": "policiesrepositdatasource",
  "targetIndexName": "policydocuments-index",
  "schedule": { "interval": "PT2H" },
  "fieldMappings": [
    { "sourceFieldName": "metadata_storage_name",
      "targetFieldName": "filename" },
    { "sourceFieldName": "metadata_storage_size",
      "targetFieldName": "filesize" },
    { "sourceFieldName": "metadata_author", "targetFieldName": "author" },
    { "sourceFieldName": "metadata_storage_last_modified",
      "targetFieldName": "modifieddate" },
    { "sourceFieldName": "metadata_storage_last_modified",
      "targetFieldName": "lastmoddate" },
    { "sourceFieldName": "metadata_content_type",
      "targetFieldName": "contenttype" },
    { "sourceFieldName": "metadata_storage_path",
      "targetFieldName": "filepath" }
  ],
  "parameters": { "base64EncodeKeys": true }
}
```

Now I can use the Azure Search explorer in the portal to verify that the documents have indeed been indexed, and the attribute values are mapped as expected.

Executing the Scenario

Running this scenario end to end requires a Web application that lets consumers register using their Microsoft credentials and submit an insurance policy request. This application would then create a contact in Dynamics CRM Online, generate a contact ID and use this value in the insurance policy request created in the list in Office 365. However, because such an application is out of scope of this article, you'll have to perform these steps manually and separately.

Log into the Dynamics CRM Online Portal and create a contact. Ensure that you capture details like the mobile number and the Microsoft Account for the customer, as these will be required later when working with the bot. The Microsoft Flow bound to

Extreme Performance Linear Scalability

For .NET Applications

Open Source (Apache 2.0)



Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing



NoSQL Database

- Schema-less JSON Documents
- Data Consistency & Replication
- Powerful SQL Support

100% Native .NET



Title	PolicyReq...	CustomerID...	PolicyType	PolicyStatus	InsuredIdentifier...	Currency	PolicyStartDa...	PolicyExpiryDat...	PolicyUniqueld
VehPolicy002	VehPolicy002	7-LKWW7GR7X	Vehicle Insurance	DocumentsSubmitted	MH04I7686	USD	7/1/2015	9/30/2016	
VehPolicy003	VehPolicy003	7-LKWW7GR7X	Vehicle Insurance	DocumentsSubmitted	MH01A7845	USD	9/1/2016	9/30/2016	

Figure 8 Insurance Policy Request in Office 365 List

the Contacts table would get triggered, invoke a stored procedure in Azure SQL Database and then insert a customer record in the CrmCustomerdata table. Note down the value of the Customer ID stored in the table for the next step.

Manually create an insurance policy request in the SharePoint custom list in Office 365. Ensure the customer ID generated for this customer in Dynamics CRM Online is captured in the request. **Figure 8** shows a list in SharePoint where the request is captured. A Microsoft Flow process bound to this list will be triggered when the record is saved, or when it's updated later during the approval process. Ensure that the policy request is replicated to the Azure SQL database PolicyRequests table.

Next, create PDF document samples and ensure they contain vehicle registration or identification numbers to be used during document search. Upload these documents to the SharePoint library that's bound to the Microsoft Flow process for document synchronization. Verify that these documents are replicated to the Azure storage account configured in the flow.

Manually run the indexer to ensure that the data synchronized in the previous steps is indexed and available to be searched. Launch the Azure Search Explorer and query for records in each of the tables and the policy documents you've uploaded and ensure they're returned in the search results.

Natural Language Utterances in the Bot App

Users of the bot on Skype or other channels can type in their requests in natural-language style, which the bot should then interpret, identify the user's intent and entities, and query Azure Search and fetch the results. For the current scenario, the bot ought to cover all possible ways its users could interact with it and pose questions on the status of their insurance policy requests, or download their approved policy documents, or query policy documents from the previous years. Here are some examples of utterances the bot should be able to interpret:

- What is the status of my insurance policy request?
- Is my policy request # VehPol0101 approved?
- Is there an update to my insurance policy request # VehPol0101?
- Show me my policy document.
- Get me the policy document.
- Get me last year's policy document for Vehicle # KA 02A 8534.
- Show me the policy document for vehicle # KA 02A 8534 issued in 2014.

- Can you schedule a site inspection visit on Tuesday next week for vehicle # KA 02A 8534?

In my next article, I'll build a bot application with natural-language support. To do so, I'll create a LUIS model that will register these natural-language utterances, train it to identify user intent in them, and determine the supporting parameters or entities with which to query the Azure Search Service.

Artifacts Available for Download

The following artifacts used to implement this scenario are available for download from the GitHub repository at bit.ly/2c0fANh:

- Azure SQL Database table creation scripts for the CrmCustomer and PolicyRequests tables.
- Stored procedures used in the preceding database tables.
- Screenshots of the schema of the Office 365 list that stores the policy requests, and that of the document library where the policy documents are uploaded.

I will be uploading other artifacts pertaining to the next article to the same location.

Wrapping Up

Organizations today deal with a variety of applications that need to work together to solve their business problems. This is often a challenge in that information is distributed across the enterprise, making it difficult to consolidate and bring within reach of consumers. Microsoft Flow helps to consolidate information from across the enterprise and Azure Search provides a flexible and rich querying engine for accessing the data. There's just one service interface to integrate with to surface all of the information, reducing the time to build and deploy your client-side applications.

In Part 2 of this article series, I'll consume the information exposed by Azure Search from a consumer-facing application, such as a bot for Skype. To do so, I'll create a LUIS model to interpret user messages from the bot and translate them into structured requests for Azure Search.

SRIKANTAN SANKARAN is a technical evangelist from the DX team in India, based in Bangaluru. He works with numerous ISVs in India and helps them architect and deploy their solutions on Microsoft Azure. Reach him at sansri@microsoft.com.

THANKS to the following Microsoft technical experts for reviewing this article: Sandeep Alur and Paul Bouwer



TECH EVENTS WITH PERSPECTIVE

6 Great Conferences
1 Great Price

Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL
DECEMBER 5-9

Visual Studio **LIVE!**

ModernApps **LIVE!**

TECHMENTOR

Office & SharePoint **LIVE!**

SQL Server **LIVE!**

APPDEV **NEW!**
TRENDS

The Ultimate Education Destination

Live! 360SM is a **unique conference** where the IT and Developer community converge to debate leading edge technologies and educate themselves on current ones. These six co-located events incorporate knowledge transfer and networking, along with expert education and training, as you create your own custom conference, mixing and matching sessions and workshops to best suit your needs.

Choose the ultimate education destination: Live! 360.

**REGISTER WITH DISCOUNT
CODE L360NOV
AND SAVE \$300!**

Must use discount code L360NOV.
Scan the QR code for more details or to register.



LIVE360EVENTS.COM

EVENT PARTNERS

Microsoft **Magenic**

PLATINUM SPONSORS

Alachisoft **HYPERGRID**

SMARTBEAR

GOLD SPONSORS

GOVERLAN

IDEA

QuickBase

SILVER SPONSOR



PRODUCED BY

105 MEDIA

Introduction to the HoloLens

Adam Tuliper

This is an exciting year for new groundbreaking devices. It's said to be the year of virtual reality (VR) and augmented reality (AR), and some very notable and highly anticipated devices have started shipping, including the HoloLens, HTC and Oculus Rift CV1. John Riccitiello, the CEO of Unity Technologies, said of AR/VR: "It's a once-in-a-generation technology that's so compelling it literally changes everything," and I couldn't agree more. HoloLens, the exciting new device from Microsoft, is capable of blending the real and virtual world.

What Is HoloLens?

HoloLens is an untethered, fully self-contained Windows 10 computer that rests comfortably on your head. It's what's known as a *mixed reality* device, a device that tries to blend the real and digital worlds. You see objects placed in the world that look and—to an extent—act like they're in the real world. In contrast, VR immerses you in an environment and you typically don't see anything around you but that virtual world. You generally aren't visually aware of the

real world outside your head-mounted display (HMD). This experience can take you flying in outer space while you sit in your office chair. And AR tries to enhance the world around you with extra data, such as markers, or heads-up information that may pertain to your location. Some AR headsets simply throw text and images on a screen overlapping whatever you're looking at.

With the HoloLens, you can bring applications and objects into the world around you that understand your environment. If you want an application pinned to the wall or in mid-air like a digital screen, as shown in **Figure 1**, no problem. Such apps stay put, even when you leave your room and come back the next day. I'm constantly leaving virtual windows open in other rooms, to be surprised when I go back days later and they're still there. And that's not all. Suppose

This article discusses:

- The HoloLens and its development environment
- The Windows Holographic platform
- Interacting with the HoloLens via gaze, gesture and voice

Technologies discussed:

HoloLens, Visual Studio, Unity, C#

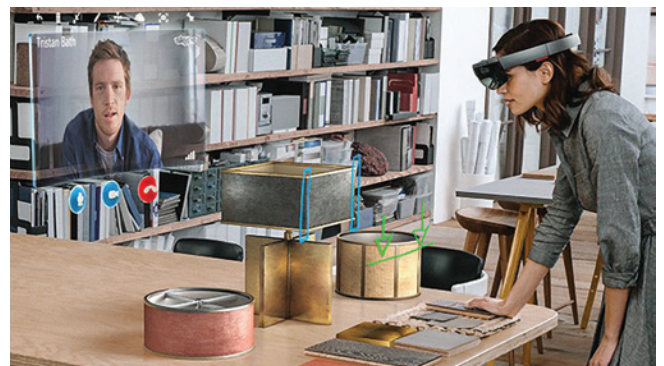


Figure 1 App on Wall

Windows 10 device families

Note that customers using a given Windows 10 device can see your product's Store listing only if you have packages which are able to run on that type of device. We recommend leaving all boxes checked unless you have a specific reason to exclude a certain Windows 10 device family.

If you remove a previously-used device family here, customers who already have your product will still be able to use it, and will get any updates you submit. However, no new customers will be able to download it on that type of Windows 10 device. [Learn more](#)

☒ Desktop

☒ Mobile

☒ Holographic

☒ Let Microsoft decide whether to make this app available to any future device families

☐ Require your permission before making this product available to any future device families.

Figure 2 Ensuring the Windows Dev Center Submission Supports the HoloLens

you want a skeleton standing in front of you in your living room that you can walk around and inspect (including climbing on your couch to look at the top of the head). Again, no problem. Drop a virtual 3D object, say a ball—referred to as a hologram—into your world and it will fall and hit your real table and stop. Move the table and the ball will fall and hit your real floor. The HoloLens understands the world around you and most are absolutely amazed the first time they try it (though I'm still waiting to be able to download Kung Fu into my brain).

In this article, I'll be covering the HoloLens development environment and the three pillars of input—gaze, gesture and voice, which represent the primary ways to interact with the HoloLens. More HoloLens features like spatial mapping, spatial audio and coordinate systems will be covered in my next article.

The Tools and SDK

There isn't a separate SDK for the HoloLens—it's simply part of the Windows 10 SDK. You just need Visual Studio 2015 with Update 3+ installed, and be sure you've checked off "Tools and Windows 10 SDK" during the install. You can go back and run the Visual Studio installer again to select these options, of course—just run the Update 3 (or higher if applicable) installer again.

The HoloLens, a Universal Windows Platform (UWP) because it's a Windows 10 device, runs UWP apps. That's right—any Windows 10 app can potentially run not only on the desktop, phone and Xbox, but also on the HoloLens. You'll find published guidance

on taking your Windows Phone and Windows Store non-UWP apps and converting them at bit.ly/2c3Hkit. To ensure that a UWP app will show up in the Windows Store, make sure you've allowed the HoloLens device family in the Windows Dev Center when your application is published, as shown in **Figure 2**. As is the case with any app on any platform, you need to ensure your application will look good on the particular devices you're targeting, as different devices have varying resolutions and memory capabilities.

The two options for developing holographic applications, the kind that can take full advantage of the HoloLens feature set, are DirectX and Unity. You can also create a plain UWP app using C# and XAML, but it will run as a 2D application on the HoloLens and you don't get the ability to drop holograms all around in space. There are project

templates in Visual Studio for holographic applications that can display holograms, as shown in **Figure 3**. Ensure prior requirements are met or you may not see the templates available. Unity (my preferred development environment for the HoloLens) has support out of the box in the HoloLens Technical Preview (HTP) builds available on its Web site at bit.ly/2bBZsNn.

The two options for developing holographic applications, the kind that can take full advantage of the HoloLens feature set, are DirectX and Unity.

At some point, however, you must touch Visual Studio, even if you're using Unity, though the development experience is quite different from using pure DirectX. Deploying to the HoloLens is done through Visual Studio and can be done via a local USB connection, a remote connection or through the emulator, as shown in **Figure 4**.

What about documentation? MSDN has long been a mainstay for developers, but years of additions made it harder to find what you needed. Now documentation is being completely redone at

docs.microsoft.com. The HoloLens team intends its documentation to be excellent and you should be able to find most things in one spot at bit.ly/1rb7i5C. If you find something missing, let the HoloLens team know. We're in a new era of making many things better at Microsoft, and the HoloLens team is helping to lead the way on good, solid documentation.

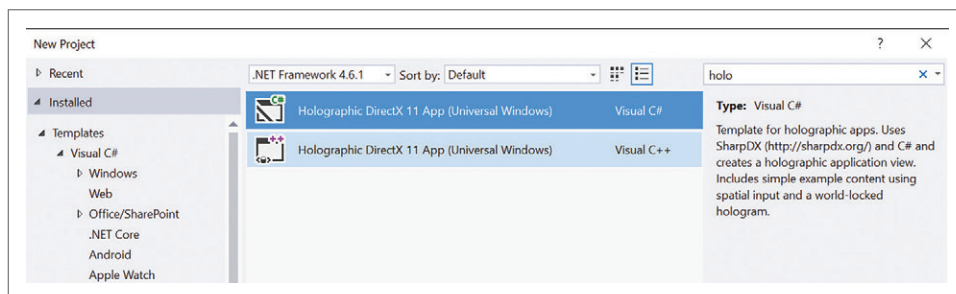


Figure 3 HoloLens DirectX Templates

The Hardware The HoloLens displays at 60 FPS, so it's extremely important to ensure your experiences run at this frame rate. You'll also want to ensure you stay within the device's memory requirements (as of this writing this means 900MB RAM available for your app). If you're new to developing graphical experiences, it's tempting to grab any art asset and place it in the scene. You may think, "I can grab a 3D model of my city and drop it into my scene and explore it!" But this would likely cause issues on any platform. Keep in mind most games implement many tricks to reduce draw calls, occlude geometry, show lower-resolution models that are further away (called level of detail, or LOD) and more, and the HoloLens is no exception. Performance recommendations are listed at bit.ly/2besJxQ. Some examples of optimization can be seen in the "Optimizing Your Games" module at bit.ly/2dBqMxw.

Technically, you don't need a HoloLens at all to develop an experience, just as you don't need a phone to develop for a phone.

The Emulator The HoloLens emulator can get you pretty far in the development process. Technically, you don't need a HoloLens at all to develop an experience, just as you don't need a phone to develop for a phone. At some point, however, you'll obviously want to test on a device. The emulator can emulate gestures, head position and voice, but it does have limitations. For one, when you're in an emulator, you won't see the real world around your holograms, as **Figure 5** shows, though you can assign a shader to your spatial mesh, which will be covered in the next article.

There are four predefined room models you can select in the emulator. Because you don't have a HoloLens on your head, the emulator should know about the room and what it looks like from a 3D standpoint. I'm jumping ahead to the next article a bit, but the HoloLens understands the space around you and is constantly updating its understanding of this space, as shown in **Figure 6**. This "behind the scenes" view of what the HoloLens sees is a room that's loaded by default in the emulator and is visible in the Windows Device Portal, which you can access for the emulator or your real HoloLens device. You can save a room model you've scanned with the HoloLens and load that into the

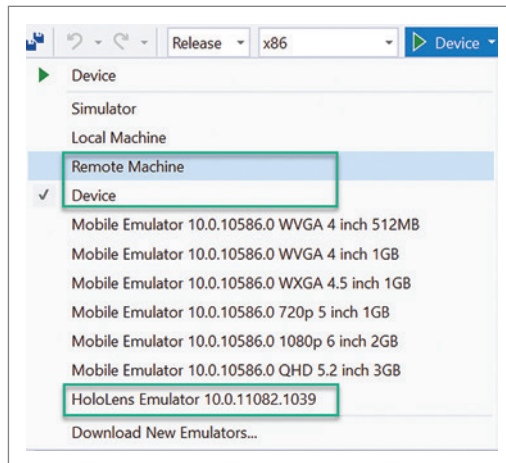


Figure 4 Different Ways to Deploy to the HoloLens

emulator—which means you can also develop multi-user experiences using a HoloLens and the emulator or multiple instances of the emulator, and load in your own custom rooms to test out.

Windows Holographic

Windows 10 contains APIs across desktop, mobile and Xbox for the Windows Holographic platform. It's through these APIs that you can interact with not only the HoloLens but other future devices, as well, including initiatives from other vendors, such as Intel's new Project Alloy untethered headset, HTC, Qualcomm, and many others. Some of the APIs for Windows Holographic can be found in

the following new namespaces (I recommend looking at these to get an idea of some of the underlying classes and structures):

- Windows.Graphics.Holographic (bit.ly/2bFbeqv)
- Windows.Perception (bit.ly/2bVNMHj)
- Windows.Perception.Spatial (bit.ly/2bwjB8h)
- Windows.Perception.Spatial.Surfaces (bit.ly/2bG9jCZ)
- Windows.UI.Input.Spatial (bit.ly/2bipB0o)

There are three pillars of input on the HoloLens—gaze, gesture and voice, and these are what I'll focus on now. If you've read any of my prior articles on game development or watched any of my online videos for Microsoft Virtual Academy, you know I'm very fond of the game engine Unity, so the code samples that follow are Unity C#-based.

Gaze is essentially where the HoloLens is, though it's easiest to consider it when it's on your head. To understand what gaze represents, let's look within the Windows.Perception.People namespace at the HeadPose class, which contains three Vector3 types (a Vector3 struct contains just x,y,z single values):

- ForwardDirection: Gets the forward direction of the HoloLens.
- Position: Gets the position of the HoloLens.
- UpDirection: Gets which direction is up for the HoloLens.

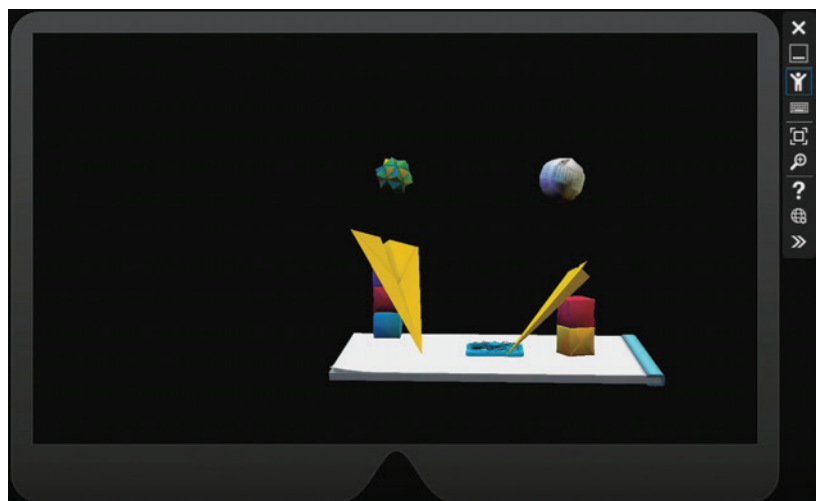


Figure 5 Using the HoloLens Emulator



ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**DEC
5-9**

**A NEW CONFERENCE FOR
SOFTWARE DEVELOPERS**

APPDEV TRENDS

ENTERPRISE FOCUSED. CODE DRIVEN.

Powering Enterprise Development

App Dev Trends, brought to you by ADTmag.com, is a new technology conference focused on the makers and maintainers of the purpose-designed software that Power organizations in virtually every industry in the world—in other words, enterprise software professionals! You Power your company. It's our job to Power you!

This event is for:

- down-in-the-trenches developers
- team leaders
- entire software development teams

Track topics include:

- Agile
- Cloud
- Mobility
- Java
- Containerization
- Continuous Integration



**REGISTER WITH DISCOUNT CODE
LEB02 AND **SAVE \$300!****



Must use discount code LEB02
for savings

Scan the QR code to register
or for more event details.

A Part of Live! 360: The Ultimate Education Destination

6 Great Conferences, 1 Great Price

Visual Studio **LIVE!**

ModernApps **LIVE!**

TECHMENTOR

Office &
SharePoint **LIVE!**

SQL Server **LIVE!**

**APPDEV
TRENDS** **NEW!**

APPDEVTRENDS.COM



PRODUCED BY
IIOS MEDIA

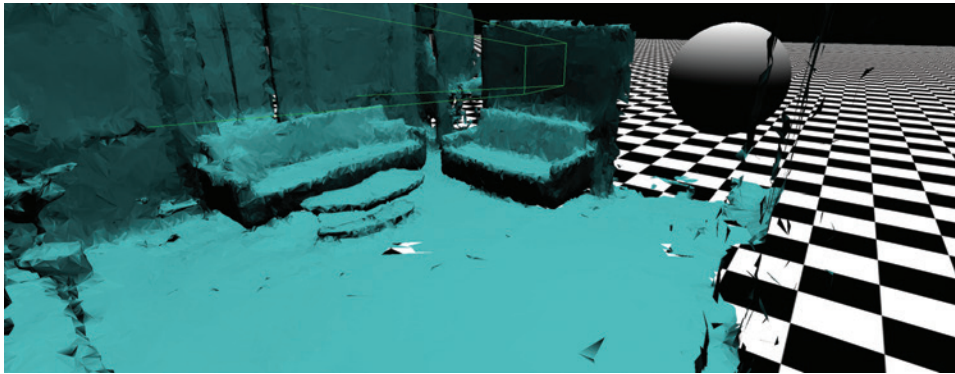


Figure 6 The Default Room Model Loaded in the Emulator Being Displayed

This information lets you determine where the user is and in which direction they're looking. When an app launches on the HoloLens, the starting position is at 0,0,0. From that position it's then easy to determine where and in which direction the user moves.

This device, while amazing,
is constrained by its size,
as is often the case
with hardware.

Let's look first at how to detect the object at which the user is looking. The requirement is that this object has a collider component on it to detect, which in Unity is trivial to add to any object. In Unity, instead of the HeadPose, you simply use the Camera.main information, which (behind the scenes) maps to the HeadPose. If

Figure 7 The Interactable Class

```
public class Interactable : MonoBehaviour
{
    // The materials we'll set to highlight.
    private Material[] defaultMaterials;
    void Awake()
    {
        // Get the materials from our renderer.
        // A material contains a shader which lights/colors object.
        defaultMaterials = GetComponent<Renderer>().materials;
    }
    void GazeEntered()
    {
        for (int i = 0; i < defaultMaterials.Length; i++)
        {
            // This assumed we're using a shader that has a Highlight
            // property, as does the Custom/SpecularHighlight one
            // from Holograms 210. This adds lighting color to it
            // to "highlight" it.
            defaultMaterials[i].SetFloat("_Highlight", .25f);
        }
    }
    void GazeExited()
    {
        for (int i = 0; i < defaultMaterials.Length; i++)
        {
            defaultMaterials[i].SetFloat("_Highlight", 0f);
        }
    }
}
```

you're used to working in Unity with the camera, nothing is different. With this information, you can shoot an invisible arrow out and find out what it hits (a common technique in 3D programming called ray casting). I just need a game object like a cube with the Interactable script (Figure 7) on it and another separate empty game object with the InteractableManager (Figure 8) on it.

I'm presenting just an overview of the main ideas here. You'll find

much more detail in Holograms 210, the Holographic Academy course on gaze concepts, including stabilizing the head position to avoid jerky movements when selecting objects (bit.ly/2b9TWIR). In short, Camera.main is the HoloLens position. You shoot your invisible ray from this position in the Camera.main.forward direction to find the first collider you hit. If you find something, all that's needed is to highlight it by setting a variable that the shader will use. For the example in Figure 7, I used the shader from Holograms 210 and just set the _Highlight value to .25 when selected. You could use the standard Unity shader, as well, and set the emission color's brightness instead, like so:

```
defaultMaterials[i].SetFloat("_EmissionColor", Color(0,0,0,.2f);
```

You need to be aware of the effect of shaders on the performance of the HoloLens. This device, while amazing, is constrained by its size, as is often the case with hardware. Nearly every experience

Figure 8 The InteractableManager Class

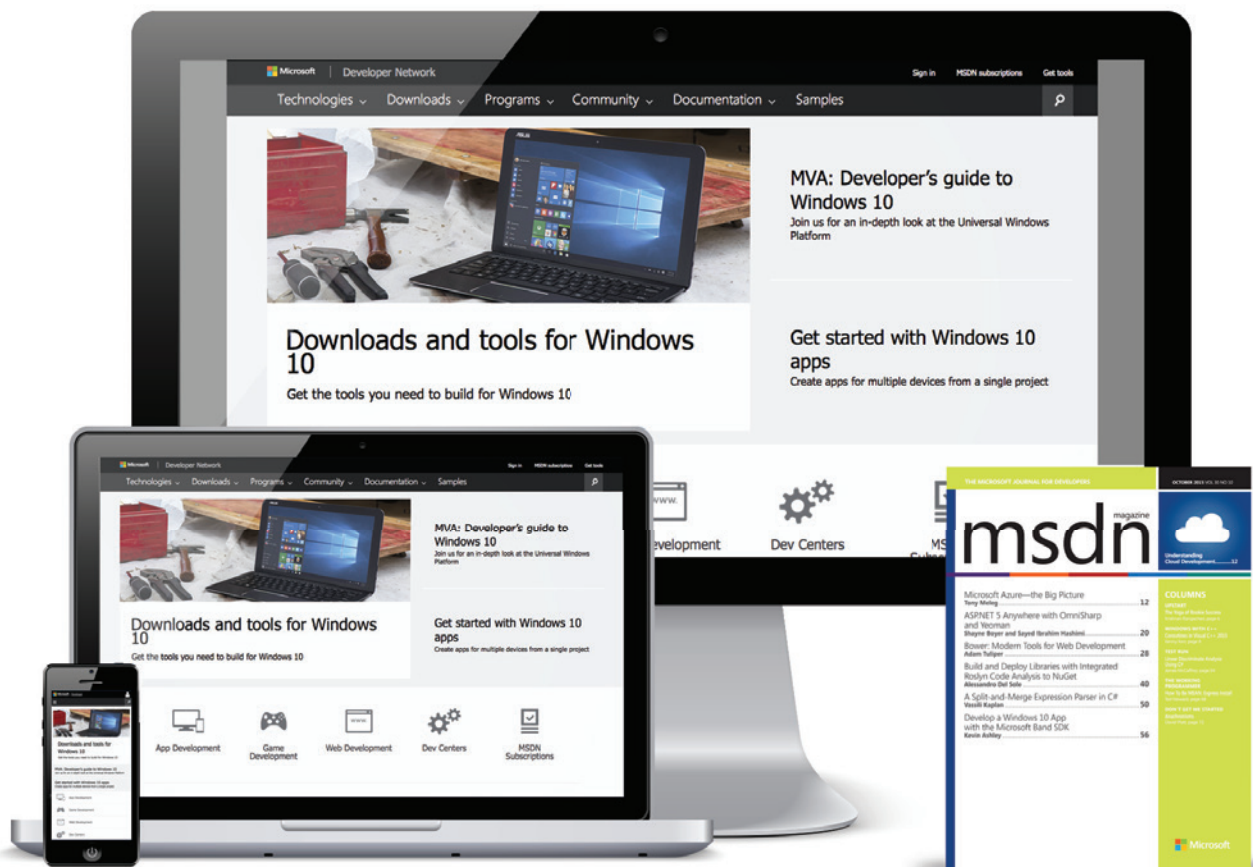
```
public class InteractableManager : MonoBehaviour
{
    private GameObject lastHit;
    // Every frame see if we are looking at a HoloGram (i.e. a game object).
    void Update()
    {
        RaycastHit hitInfo;
        if (Physics.Raycast(
            Camera.main.transform.position,
            Camera.main.transform.forward,
            out hitInfo,
            10.0f,
            Physics.DefaultRaycastLayers))
        {
            // The game object we've hit.
            var tempGO = hitInfo.collider.gameObject;

            // See if this object contains our Interactable class.
            if(tempGO.GetComponent<Interactable>() != null)
            {
                lastHit = tempGO;
                // Loosely coupled way to call a method in Unity on our g.o.
                lastHit.SendMessage("GazeEntered");
            }
        }
        else
        {
            // No object detected in Gaze, lets deselect the last one.
            if(lastHit!=null)
            {
                lastHit.SendMessage("GazeExited");
                lastHit = null;
            }
        }
    }
}
```

msdn

magazine

Where you need us most.



Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!
360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com



Figure 9 Displaying a Directional Indicator

and game made today has time allocated for optimization, and the HoloLens is no different. When developing for the HoloLens with Unity, the first thing you'll want to do is swap out the shaders for the optimized variants located in the HoloToolkit at bit.ly/2b06cH2.

The HoloToolkit contains many helper functions and objects that can greatly aid in HoloLens development, including cursors, gaze stabilizers, gesture and hand managers, plane detectors, follow-me scripts, examples for sharing across multiple HoloLens, and much more. The majority of this functionality is in the HoloToolkit-Unity at bit.ly/2b08XrT, so be sure you look in the correct repository and not just at the basic HoloToolkit at bit.ly/2bPCbas, which, however, is useful in its own right and contains various standalone sample projects and other library code that the HoloToolkit-Unity utilizes.

Because you have
access to the camera position
and any game object position,
it's easy to figure out where a
hologram is compared to where
the user is looking.

When you have objects that aren't in the viewable area, be it a game on the screen, VR, AR or mixed reality, it's useful to provide a small indicator to tell the user in what direction they should look for an object if they start to look away, as shown in **Figure 9**. This is a very common technique in many video games, and it's incredibly useful in the mixed-reality world.

The code in **Figure 10** can be used to determine if a particular hologram is visible.

Once you know if an object isn't visible (or is just barely visible), you can figure out in which direction to show a directional indicator. Because you have access to the camera position and any game object position, it's easy to figure out where a hologram is compared to where the user is looking. To get a directional vector representing an arrow from the camera to a particular game object is a matter of subtracting one location from the other and normalizing it to

Figure 10 Determining If a Hologram Is Visible

```
[Tooltip("Allowable percentage (to 30%) inside the holographic frame to  
continue to show a directional indicator.")]  
[Range(-0.3f, 0.3f)]  
public float TitleSafeFactor = 0.1f;  
  
// Determine if sgame object is visible within a certain percentage.  
// The Viewport is 0,0 to 1,1 (bottom left of screen to top right).  
private bool IsTargetVisible()  
{  
    // This will return true if the target's mesh is within the Main  
    // Camera's view frustums.  
    Vector3 targetViewportPosition =  
        Camera.main.WorldToViewportPoint(gameObject.transform.position);  
    return (targetViewportPosition.x > TitleSafeFactor &&  
        targetViewportPosition.x < 1 - TitleSafeFactor &&  
        targetViewportPosition.y > TitleSafeFactor &&  
        targetViewportPosition.y < 1 - TitleSafeFactor &&  
        targetViewportPosition.z > 0);  
}
```

make it one-based (it's easiest to use a normalized vector in many calculations), like so:

```
Vector3 camToObjectDirection =  
    gameObject.transform.position - Camera.main.transform.position;  
camToObjectDirection.Normalize();
```

Showing directional indicators does require a couple other things to be set up, but this is the gist of the functionality. Once you know the direction from the HoloLens (that is, the camera) to the hologram, you can display an indicator on the screen. You can check out this feature in more detail at bit.ly/2bh0Hz7.

Gesture is the next way of processing input to the HoloLens. Gestures are performed with the hand, controller or voice commands. Hand gestures are scanned for via one of the cameras on the front, within what's called the gesture frame, an expanse that extends the viewing area on all sides of the hologram, allowing you to keep your hand closer to your body rather than holding it way out every time. You typically don't tap on holograms, but instead tap and determine what the user is looking at via gaze.

The HoloLens supports two categories of gestures. The first includes discrete gestures, such as a single quick action like the air tap shown in **Figure 11**, a click with the included Bluetooth clicker, saying the command "Select," or even a double tap. The second type involves continuous gestures, which happen over time and are generally triggered when you press and hold and then move your hands. Continuous gestures provide support for navigation and manipulation. Navigation gestures give you a starting position of 0,0,0—which is wherever you start the gesture—and then are constrained to a virtual cube in space that allows you to move from -1 to 1 on any axis



Figure 11 An Air Tap

within that cube to provide (among other use cases) smooth rotation around one or multiple axes or smooth scrolling. Manipulation gestures, rather than constraining to a virtual cube, allow a 1:1 movement between hands and holograms—think of painting virtually or nice fluid motions to position holograms around your environment; you get world-relative positional information in x,y,z.

As a developer you can, of course, hook into the gestures, like air tap, hold, navigation and manipulation. Currently there's no support for custom gestures, but you do have access to quite a bit more data via the aforementioned gestures. You can also track

Figure 12 The GestureManager Class

```
public class GestureManager : MonoBehaviour
{
    private GestureRecognizer gestureRecognizer;

    void Start()
    {
        gestureRecognizer = new GestureRecognizer();
        // We can register here for more than just tap (ex Navigation).
        gestureRecognizer.SetRecognizableGestures(GestureSettings.Tap);

        gestureRecognizer.TappedEvent += (source, tapCount, ray) =>
        {
            // A tap has been detected.
            // Raycast using the provided ray and get the hit game
            // object and perform some action on it as was done previously.
        };

        gestureRecognizer.StartCapturingGestures();
    }

    void OnDestroy()
    {
        gestureRecognizer.StopCapturingGestures();
    }
}
```

Figure 13 Adding Keywords and Delegate Code

```
public class SpeechManager : MonoBehaviour
{
    KeywordRecognizer keywordRecognizer;
    Dictionary<string, System.Action> keywords =
        new Dictionary<string, System.Action>();

    void Start()
    {
        keywords.Add("Reset level", () =>
        {
            // Call the OnReset method on every descendant object to reset their state.
            this.BroadcastMessage("OnReset");

            // We could also do a full-level reload via
            // SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        });

        // Tell the KeywordRecognizer about our keywords.
        keywordRecognizer = new KeywordRecognizer(keywords.Keys.ToArray());

        // Register a callback for the KeywordRecognizer and start recognizing!
        keywordRecognizer.OnPhraseRecognized += KeywordRecognizer_OnPhraseRecognized;
        keywordRecognizer.Start();
    }

    private void KeywordRecognizer_
    OnPhraseRecognized(PhraseRecognizedEventArgs args)
    {
        System.Action keywordAction;
        if (keywords.TryGetValue(args.text, out keywordAction))
        {
            keywordAction.Invoke();
        }
    }
}
```

hand position (when in the ready state or pressed state) in x,y,z and understand when the hands are in or leaving the gesture frame, which can provide helpful feedback to the user when their hands are about to leave the gesture frame.

You can add a GestureManager class like the one in **Figure 12** to register for the tapped event, which will let you know, for example, when the user has performed a tap gesture via the Select voice command, a controller or a hand (regardless of where they tapped). You then need to determine what was being looked at when the user tapped, just as what was done previously. It's important to note that the TappedEvent in **Figure 12** is passed in a ray that represents where the head was when the event happened. In the case of fast head movement, you want to ensure you raycast from where the gaze was when the user taps, not where it is when the event arrives, which may be slightly off of the original location.

The HoloLens opens up a new era in how you can experience the world around you, mixing both reality and the virtual world.

Voice commands on the HoloLens are supported on a system level through Cortana, but also share the same speech engine as all other UWP apps, and thus require the Microphone device capability in the project's package.appxmanifest file. The audio processing is hardware-accelerated and provides 16khz to 48khz 24-bit audio streams of the user's voice and ambient environment. Built-in Cortana commands include "Select," "Hey Cortana <command>," and in-app commands contain "Select" (instead of air-tapping) and "Place."

Responding to custom voice commands is possible and straightforward. Just add your keywords and delegate code to execute and start listening, as shown in **Figure 13**.

For those looking to do speech-to-text, dictation is available, as well, using the DictationManager class to get DictationHypothesis, DictationResult, DictationComplete and DictationError events. Do note this requires WiFi connectivity. My coworker Jared Bienz has added some nice support for text-to-speech, as well, which you can find in the HoloToolkit for Unity.

Wrapping Up

The HoloLens opens up a new era in how you can experience the world around you, mixing both reality and the virtual world. Get started at HoloLens.com and keep an eye there for announcements. Stay tuned for my next article, which will discuss my favorite HoloLens feature, spatial mapping. ■

ADAM TULIPER is a senior technical evangelist with Microsoft living in sunny SoCal. He's a Web dev/game dev Pluralsight.com author, and all-around tech lover. Find him on Twitter: @AdamTuliper or at channel9.msdn.com/Blogs/AdamsGarage.

THANKS to the following Microsoft technical expert for reviewing this article:
Jackson Fields

Hidden Disposables

Artak Mkrtchyan

Disposable types are great, as they let you free up resources in a deterministic way. However, there are situations where developers work with disposable types without even realizing it. The use of creational design patterns is an example of a situation where usage of a disposable type might not be obvious, which can lead to an object not getting disposed. This article will show ways to handle the problem. I'll start by reviewing some of the creational design patterns.

Creational Design Patterns

A great benefit of the creational design patterns is that they abstract away from the actual implementations and “talk” in the interface language. They deal with object creation mechanisms to create objects that are suitable to the solution. In comparison with basic object creation, creational design patterns improve several aspects of the object creation process. Here are two well-known benefits of creational design patterns:

- **Abstraction:** They abstract the object type being created, so the caller doesn't know what the actual object being returned is—they're aware of only the interface.
- **Creation internals:** They encapsulate the knowledge about the specific type instance creation.

Next, I'll give a brief overview of two well-known creational design patterns.

This article discusses:

- The importance of disposable types and the Dispose design pattern
- Object creation abstractions and creational design patterns
- Non-obvious or hidden usage of disposable types

Technologies discussed:

IDisposable interface, Creational Design Patterns, Object Creation Abstractions

The Factory Method Design Pattern The Factory Method design pattern is one of my favorites and I use it a lot in my day-to-day work. This pattern uses factory methods to deal with the problem of creating objects without specifying the exact class of the object that's created. Rather than calling a class constructor directly, you call a factory method to create the object. The factory method returns an abstraction (interface or a base class), which child classes implement. **Figure 1** shows the Unified Modeling Language (UML) diagram for this pattern.

In **Figure 1**, the ConcreteProduct is a specific type of the IProduct abstraction/interface. Similarly, the ConcreteCreator is a specific implementation of the ICreator interface.

The client of this pattern uses an ICreator instance, and will be calling its Create method to get a new instance of IProduct, without knowing which actual product has been returned.

The Abstract Factory Design Pattern The goal of the Abstract Factory design pattern is to provide an interface for creating families of related or dependent objects without specifying concrete implementations.

This shelters the client code from the hassle of object creation by having the client ask the factory object to create an object of the desired abstract type and to return an abstract pointer to the object back to the client. In particular, this means the client code has no knowledge about the concrete type. It deals only with an abstract type.

Adding support for new concrete types is handled by creating new factory types and modifying the client code to use a different factory type as needed. In most cases, this is a one-line code change. This obviously simplifies handling changes, as the client code doesn't need to change to accommodate the new factory type. **Figure 2** shows the UML diagram for the Abstract Factory design pattern.

From a client perspective, the usage of the Abstract Factory is represented with the following piece of code:

```
IAbstractFactory factory = new ConcreteFactory1();  
IProductA product = factory.CreateProductA();
```

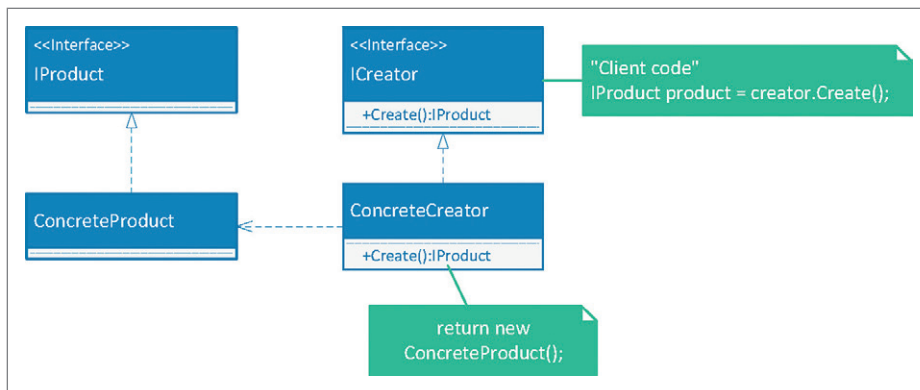


Figure 1 Factory Method Design Pattern

The client is free to modify the actual factory implementation to control the type of product created behind the scenes and that will have absolutely no impact on the code.

This code is just a sample; in properly structured code, the factory instantiation itself would probably be abstracted away—with a factory method pattern as an example.

The Problem

In both of the design pattern examples, there was a factory involved. A factory is the actual method/procedure, which returns a constructed type reference through an abstraction in response to the client's call.

Technically, you can use a factory to create an object, wherever an abstraction exists, as shown in **Figure 3**.

The factory handles the choice between different available implementations, based on the factors involved.

According to the Dependency Inversion principle:

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

This, technically speaking, means that on every level of a dependency chain, the dependency should be replaced by an abstraction. In addition, the creation of these abstractions can—and in many cases should be—handled through factories.

All this emphasizes how important factories are in day-to-day coding. However, they're actually hiding the problem: disposable types. Before getting into those details, I'll first talk about the IDisposable interface and the Dispose Design pattern.

Dispose Design Pattern

All programs acquire resources such as memory, file handles and database connections during their execution. Developers have to be careful when using such resources because the resources must be released after they've been acquired and used.

The Common Language Runtime (CLR) provides support for automatic memory management through the garbage collector (GC). You don't have to clean up Managed Memory explicitly because the GC will do that

automatically. Unfortunately, there are other types of resources (referred to as Unmanaged resources) that still need to be explicitly released. The GC isn't designed to handle these types of resources, so it's the developer's responsibility to release those.

Still, the CLR helps developers deal with unmanaged resources. The System.Object type defines a public virtual method, called

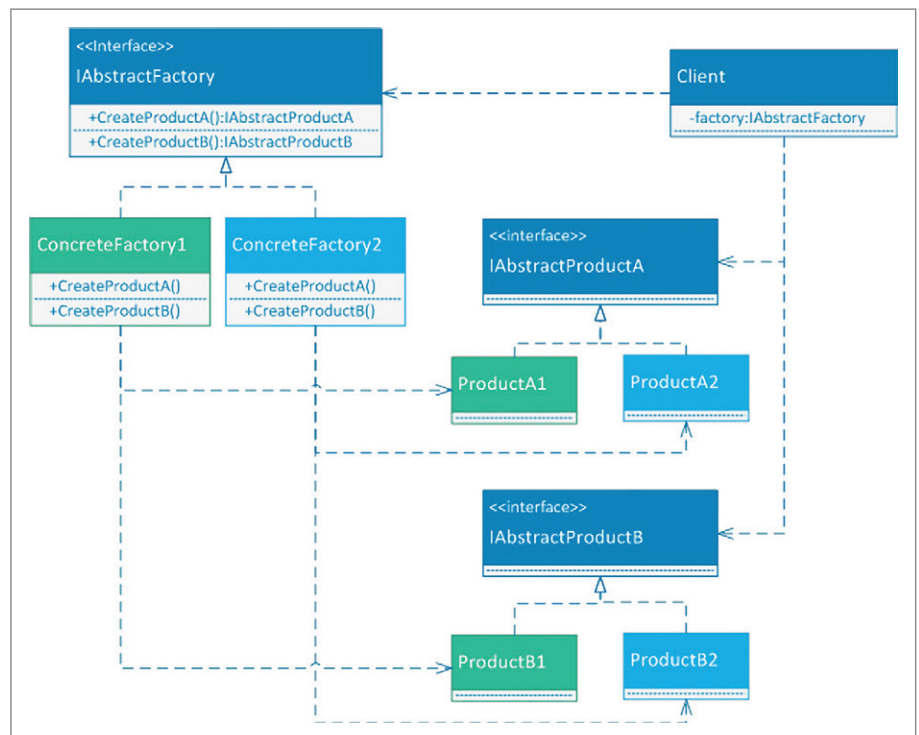


Figure 2 Abstract Factory Design Pattern

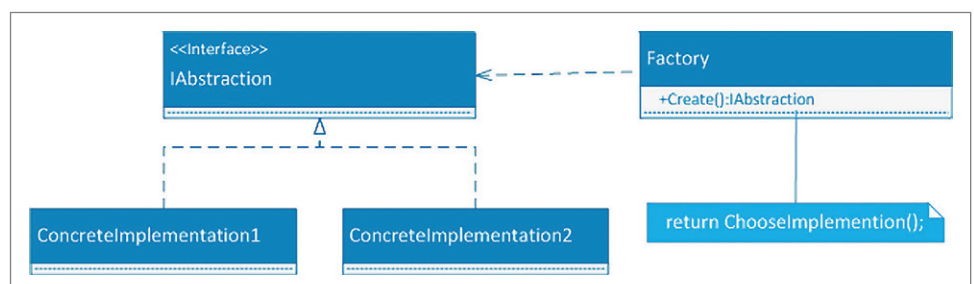


Figure 3 Example of Simple Abstraction and Its Usage

Figure 4 Implementation of the Dispose Design Pattern

```
public class DisposableType : IDisposable {
    ~DisposableType() {
        this.Dispose(false);
    }

    public void Dispose() {
        this.Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing) {
        if (disposing) {
            // Dispose of all the managed resources here
        }

        // Dispose of all the unmanaged resources here
    }
}
```

Finalize, which is called by the GC before the object's memory is reclaimed. The Finalize method usually is referred to as finalizer. You can override the method to clean up additional unmanaged resources used by the object.

This mechanism, however, has some drawbacks due to certain aspects of the GC execution.

The finalizer is called when the GC detects that an object is eligible for collection. This happens at an undetermined period after the object isn't needed anymore.

When the GC needs to call the finalizer, it has to postpone the actual memory collection to the next round of garbage collection. This postpones the object's memory collection even longer. That's where the System.IDisposable interface comes in. The Microsoft .NET Framework provides the IDisposable interface that you need to implement to provide the developer a mechanism for manually releasing unmanaged resources. Types that implement this interface are called disposable types. The IDisposable interface defines just one parameter-less method, called Dispose. Dispose should be called to immediately release any unmanaged resources it references as soon as the object isn't needed.

You might ask, "Why should I call Dispose myself when I know that GC will eventually handle it for me?" The answer requires a separate article, which also touches on aspects of the GC's execution impact on performance. This is beyond the scope of this article, so I'll move on.

There are certain rules to follow when deciding whether a type should be disposable. The rule of thumb is this: If an object of a given type is going to reference an unmanaged resource or other disposable objects, then it should also be disposable.

The Dispose pattern defines a specific implementation for the IDisposable interface. It requires two Dispose methods to be implemented: one public without parameters (defined by the IDisposable interface) and the other one a protected virtual with a single Boolean parameter. Obviously, if the type is going to be sealed, the protected virtual should be replaced by private.

The Boolean parameter indicates the way in which the dispose method is being called. The public method calls the protected one with a parameter value "true." Similarly, the overloads of the Dispose(bool) method in the class hierarchy should call base.Dispose(true).

The Dispose pattern implementation also requires the Finalize method to be overloaded. This is done to cover scenarios where a developer forgets to call the Dispose method after the object isn't needed anymore. Because the finalizer is being called by the GC, the referenced managed resources might already (or will) be cleaned up, so you should handle the release of unmanaged resources only when the Dispose(bool) method is called from the finalizer.

Switching back to the main topic, the problem comes into play when you deal with disposable objects when used with creational design patterns.

Imagine a scenario where one of the concrete types implementing the abstraction also implements the IDisposable interface. Let's assume it's the ConcreteImplementation2 in my example, as shown in **Figure 5**.

Note that the IAbstraction interface itself does not inherit from the IDisposable.

Now look at the client code, where the abstraction is going to be used. As the IAbstraction interface hasn't changed, the client won't even care about any potential changes behind the scenes. Naturally, the client won't guess that he's been given an object, of which he's now responsible to dispose. The reality is that an IDisposable instance isn't really expected there and in many cases, those objects never get disposed explicitly by the client code.

The hope is that the actual implementation of the ConcreteImplementation2 implements the Dispose Design Pattern, which isn't always the case.

It's now obvious that the simplest mechanism to handle a case where the returned IAbstraction instance also implements IDisposable interface would be to introduce an explicit check in the client code, as shown here:

```
IAbstraction abstraction = factory.Create();
try {
    // Operations with abstraction go here
}
finally {
    if (abstraction is IDisposable)
        (abstraction as IDisposable).Dispose();
}
```

This, however, soon becomes a tedious procedure.

Unfortunately, a using block can't be used with IAbstraction, as it doesn't extend IDisposable explicitly. So I came up with a helper class, which wraps the logic in the finally block and lets you use the using block, as well. **Figure 6** shows the full code of the class and also provides a sample usage.

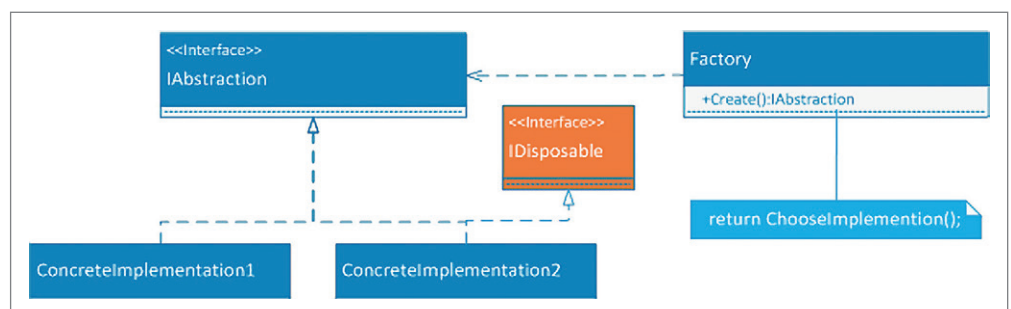


Figure 5 Abstraction with an IDisposable Implementation

Figure 6 PotentialDisposable Type and Its Usage

```
public sealed class PotentialDisposable<T> : IDisposable where T : class
{
    private readonly T instance;

    public T Instance { get { return this.instance; } }

    public PotentialDisposable(T instance) {
        if (instance == null) {
            throw new ArgumentNullException("instance");
        }

        this.instance = instance;
    }

    public void Dispose() {
        IDisposable disposableInstance = this.Instance as IDisposable;
        if (disposableInstance != null) {
            disposableInstance.Dispose();
        }
    }
}

The client code:
IAbstraction abstraction = factory.Create();
using (PotentialDisposable<IAbstraction> wrappedInstance =
    new PotentialDisposable<IAbstraction>(abstraction)) {
    // Operations with abstraction wrappedInstance.Instance go here
}
```

As you can see in the “The client code” part of **Figure 6**, using the `PotentialDisposable<T>` class reduced the client code to just a couple of lines with a `using` block.

You could argue that you could just update the `IAbstraction` interface and make it `IDisposable`. This may be the preferred solution in some situations, but not others.

In a situation where you own the `IAbstraction` interface and it makes sense for `IAbstraction` to extend `IDisposable`, you should do that. Actually, a good example of this would be the `System.IO.Stream` abstract class. The class actually implements the `IDisposable` interface, but it has no actual logic defined. The reason is that the writers of the class knew that most of the child classes will have some type of disposable members.

Another situation: When you do own the `IAbstraction` interface, but it doesn't make sense for it to extend `IDisposable`, as most of the implementations of it are not disposable. Think about an `ICustomCollection` interface as an example. You have several in-memory implementations and suddenly you need to add some database-backed implementation, which will be the only disposable implementation.

The final situation would be when you don't own `IAbstraction` interface—so you don't have control over it. Consider an example of `ICollection`, which is backed by a database.

Wrapping Up

Whether or not the abstraction you get is through a factory method, it's important to keep disposables in mind when writing your client code. Using this simple helper class is one way to ensure your code is being as efficient as possible when dealing with disposable objects. ■

ARTAK MKRTRYAN is a senior software engineer living in Redmond, Wash. He loves coding as much as he loves fishing.

THANKS to the following Microsoft technical expert for reviewing this article:
Paul Brambilla

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION

1. Publication Title: *MSDN Magazine*
2. Publication Number: 1528-4859
3. Filing Date: September 30, 2016
4. Frequency of Issue: Monthly with a special issue in October.
5. Number of Issues Published Annually: 13
6. Annual Subscription Price: US \$35, International \$60
7. Complete Mailing Address of Known Office of Publication: 9201 Oakdale Ave., Ste. 101, Chatsworth, CA 91311
8. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher: Same as above.
9. Full Name and Complete Mailing Address of Publisher, Editor, and Managing Editor:
Henry Allain, President, 4 Venture, Suite 150, Irvine, CA 92618
Michael Desmond, Editor-in-Chief, 8609 Westwood Center Dr., Suite 500, Vienna, VA 22182
Wendy Hernandez, Group Managing Editor, 4 Venture, Ste. 150, Irvine, CA 92618
10. Owner(s): 1105 Media, Inc. dba: 101 Communications LLC, 9201 Oakdale Ave, Ste. 101, Chatsworth, CA 91311. Listing of shareholders in 1105 Media, Inc.
11. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or more of the Total Amount of Bonds, Mortgages or Other Securities:
Nautic Partners V, L.P., 50 Kennedy Plaza, 12th Fl., Providence, RI 02903
Kennedy Plaza Partners III, LLC, 50 Kennedy Plaza, 12th Fl., Providence, RI 02903
Alta Communications IX, L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
Alta Communications IX, B-L.P., 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
Alta Communications IX, Associates LLC, 1000 Winter Street, South Entrance, Suite 3500, Waltham, MA 02451
12. The tax status has not changed during the preceding 12 months.
13. Publication Title: *MSDN Magazine*
14. Issue date for Circulation Data Below: September 2016
15. Extent & Nature of Circulation:

	Average No. Copies Each Month During Preceding 12 Months	No. Copies of Single Issue Published Nearest to Filing Date
a. Total Number of Copies (Net Press Run)	83,146	99,364
b. Legitimate Paid/and or Requested Distribution		
1. Outside County Paid/Requested Mail Subscriptions Stated on PS Form 3541	66,057	66,402
2. In-County Paid/Requested Mail Subscriptions Stated on PS Form 3541	0	0
3. Sales Through Dealers and Carriers, Street Vendors, Counter Sales, and Other Paid or Requested Distribution Outside USPS®	3,316	3,641
4. Requested Copies Distributed by Other Mail Classes Through the USPS®	0	0
c. Total Paid and/or Requested Circulation	69,373	70,043
Nonrequested Distribution		
1. Outside County Nonrequested Copies Stated on PS Form 3541	11,112	10,367
2. In-County Nonrequested Copies Distribution Stated on PS Form 3541	0	0
3. Nonrequested Copies Distribution Through the USPS by Other Classes of Mail	0	0
4. Nonrequested Copies Distributed Outside the Mail	2,517	18,839
e. Total Nonrequested Distribution	13,629	29,206
f. Total Distribution	83,002	99,249
g. Copies not Distributed	144	115
h. Total	83,146	99,364
i. Percent paid and/or Requested Circulation	83.58%	70.57%

16. Electronic Copy Circulation
 - a. Requested and Paid Electronic Copies
 - b. Total Requested and Paid Print Copies (Line 15c) + Requested/Paid Electronic Copies
 - c. Total Requested Copy Distribution (Line 15f) + Requested/Paid Electronic Copies (Line 16a)
 - d. Percent Paid and/or Requested Circulation (Both print & Electronic Copies) (16b divided by 16c x 100)

☒ I certify that 50% of all my distributed copies (electronic and paid print are legitimate request or paid copies.
17. Publication of Statement of Ownership for a Requester Publication is required and will be printed in the November 2016 issue of this publication.
18. I certify that all information furnished on this form is true and complete:
Peter B. Weller, Manager, Print Production



Solving Sudoku Using Combinatorial Evolution

A combinatorial optimization problem is one where the goal is to arrange a set of discrete items into a particular order. A Sudoku puzzle is an example of a combinatorial optimization problem. In this article, I show you how to write a program to solve difficult Sudoku problems, using a technique I call combinatorial evolution.

The demo sets up a non-easy Sudoku problem (I'll explain what I mean by non-easy shortly). In Sudoku, there are several constraints. Each row of the 9x9 solution grid must contain the numbers 1 through 9, with no duplicates. Each column must contain the numbers 1 through 9. Each 3x3 sub-grid must contain the numbers 1 through 9. And the solution grid cannot change any of the starting values in the problem grid.

The demo problem is shown in **Figure 1**. Some Sudoku problems can be solved using a brute force algorithm where you examine each empty cell in the grid and check to see which values can be legally placed there by checking the row, column and sub-grid constraints. If only one value is possible, then that value is placed in the empty cell. The process is continued until all empty cells have been assigned. These non-easy type of Sudoku problems are the ones usually found in newspapers and magazines.

The demo problem is non-easy because the brute-force algorithm doesn't work. Suppose you scan the problem grid from left to right, and then top to bottom. The empty cell at (0, 0) can be one of (1, 3, 5, 9). The empty cell at (0, 1) can be one of (1, 3, 9). The next empty cell at (0, 4) can only be a 3, so you can place a 3 there and continue. However, using this approach, after placing nine values, you'd get stuck because all remaining cells could be two or more values.

To get an idea of what combinatorial evolution optimization is, take a look at the screenshot in **Figure 2**. Combinatorial evolution optimization uses ideas from several bio-inspired algorithms. The algorithm maintains a collection of virtual organisms. Each organism represents a possible solution to the problem. Combinatorial evolution is an iterative process. Each iteration is called an epoch. In each epoch, every organism attempts to find a better solution by examining a new possible solution.

After all organisms have had a chance to improve, two good organism-solutions are selected and used to give birth to a new organism, which replaces a poor solution. So the population of organisms evolves over time. If an optimal solution isn't found after some maxEpochs time, the algorithm is restarted by killing all the organisms and creating a new population.

The demo sets up 200 virtual organisms and a time limit of 5,000 epochs. These values were found by using a little bit of trial and error. Combinatorial evolution doesn't guarantee an optimal solution will be found, so a limit of 20 restarts was set to prevent a possible infinite loop.

The demo program found an optimal solution after three restarts, which took approximately 8 seconds running on my desktop machine. During the demo run, the program displayed a measure of error of the best organism. I'll explain how error is defined and calculated when I present the relevant code. However, notice that the algorithm tends to get a very good solution (error = 2) very quickly, but then gets stuck. The restart process is one mechanism to combat this characteristic, and is a common technique in many optimization algorithms.

The Demo Program

To create the demo program, I launched Visual Studio, clicked on File | New | Project and selected the C# Console Application option. I named the project SudokuEvo. The demo program has no significant .NET dependencies so any version of Visual Studio will work. After the template code loaded, in the Solution Explorer window, I right-clicked on file Program.cs and renamed it to SudokuEvoProgram.cs and allowed Visual Studio to automatically rename class Program for me.

At the top of the editor window, I deleted all unnecessary using statements, leaving just references to the System and Collections. Generic namespaces. The overall structure of the program, with a few minor edits, is shown in **Figure 3**. The demo program is too long to present in its entirety in this article, but the complete demo source code is available in the accompanying download.

		6	2				8		7	1	6	2	3	5	9	8	4
		8	9	7					5	2	8	9	7	4	3	1	6
		4	8	1		5			3	9	4	8	1	6	5	2	7
				6					8	4	5	1	6	3	7	9	2
	7							3	2	7	1	4	8	9	6	3	5
6				5					6	3	9	7	5	2	8	4	1
		2		4	7	1			9	8	2	6	4	7	1	5	3
		3		2	8	4			1	6	3	5	2	8	4	7	9
	5				1	2			4	5	7	3	9	1	2	6	8
Problem									Solution								

Figure 1 A Non-Easy Sudoku Problem

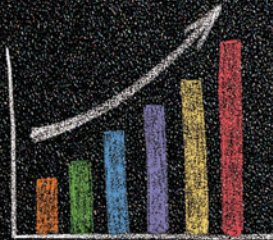
Code download available at msdn.com/magazine/1116magcode.

Spreadsheets Made Easy.



Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.



Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Windows
Forms



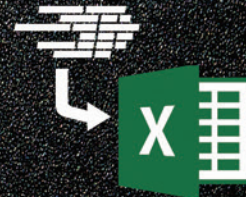
Silverlight



WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com



SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

The demo program isn't as complicated as it might first appear, because many of the methods are short helper methods. The demo has two classes. The main class has all the code logic implemented as static methods. The Organism class defines a possible solution to the target Sudoku problem.

Each Organism object has a type, which can be 0 for a "worker" organism, or 1 for an "explorer" organism. The field named matrix is an integer array-of-arrays-style matrix that represents a possible solution. Each possible solution has an error, where an error value of 0 means that no constraints are violated and, therefore, the matrix field holds an optimal solution. Each Organism object has an age field to control whether the organism dies in each epoch.

The demo program sets up and displays the Sudoku problem using these statements:

```
int[][] problem = new int[9][];
problem[0] = new int[] { 0, 0, 6, 2, 0, 0, 0, 8, 0 };
...
problem[8] = new int[] { 0, 5, 0, 0, 0, 1, 2, 0, 0 };
DisplayMatrix(problem);
```

```
file:///C:/SudokuEvo/bin/Debug/SudokuEvo.EXE
Begin solving Sudoku using combinatorial evolution

The problem is:
-- 6 2 -- -- 8 --
-- 8 9 7 -- -- --
-- 4 8 1 -- 5 --
-- 7 -- -- 6 -- -- 2
6 -- -- 5 -- -- 3 --
-- 2 -- 4 7 1 -- --
-- 3 -- 2 8 4 -- --
-- 5 -- -- 1 2 -- --

Setting numOrganisms = 200
Setting maxEpochs = 5000
Setting maxRestarts = 20

seed = 0
epoch = 0 best error = 31
epoch = 1000 best error = 2
epoch = 2000 best error = 2
epoch = 3000 best error = 2
epoch = 4000 best error = 2

seed = 1
epoch = 0 best error = 35
epoch = 1000 best error = 4
epoch = 2000 best error = 4
epoch = 3000 best error = 2
epoch = 4000 best error = 2

seed = 2
epoch = 0 best error = 32
epoch = 1000 best error = 4
epoch = 2000 best error = 2
epoch = 3000 best error = 2
epoch = 4000 best error = 2

seed = 3
epoch = 0 best error = 29
epoch = 1000 best error = 2

Best solution found:
7 1 6 2 3 5 9 8 4
5 2 8 9 7 4 3 1 6
3 9 4 8 1 6 5 2 7

8 4 5 1 6 3 7 9 2
2 7 1 4 8 9 6 3 5
6 3 9 7 5 2 8 4 1

9 8 2 6 4 7 1 5 3
1 6 3 5 2 8 4 7 9
4 5 7 3 9 1 2 6 8

Success

End Sudoku using combinatorial evolution
```

Figure 2 Solving Sudoku Using Combinatorial Evolution

Notice that 0 values are used to indicate an empty cell. This manual, hardcoded approach is pretty tedious and in most realistic combinatorial optimization problems you'd read problem data from a text file.

The Sudoku problem is tackled using these statements:

```
int numOrganisms = 200;
int maxEpochs = 5000;
int maxRestarts = 20;
int[][] soln = Solve(problem, numOrganisms, maxEpochs, maxRestarts);
Console.WriteLine("Best solution found: ");
DisplayMatrix(soln);
```

Method Solve is mostly a wrapper around method SolveEvo, which does most of the work. This is a common design pattern in combinatorial optimization—one low-level solver method attempts to find an optimal solution, and that method is wrapped by a high-level solver method that performs restarts.

The combinatorial evolution algorithm isn't guaranteed to find an optimal solution (that is, a solution that has no constraint errors), so the demo program checks to determine if the best solution found is optimal:

```
int err = Error(soln);
if (err == 0)
    Console.WriteLine("Success");
else
    Console.WriteLine("Did not find optimal solution");
```

Matrix Initialization and Error

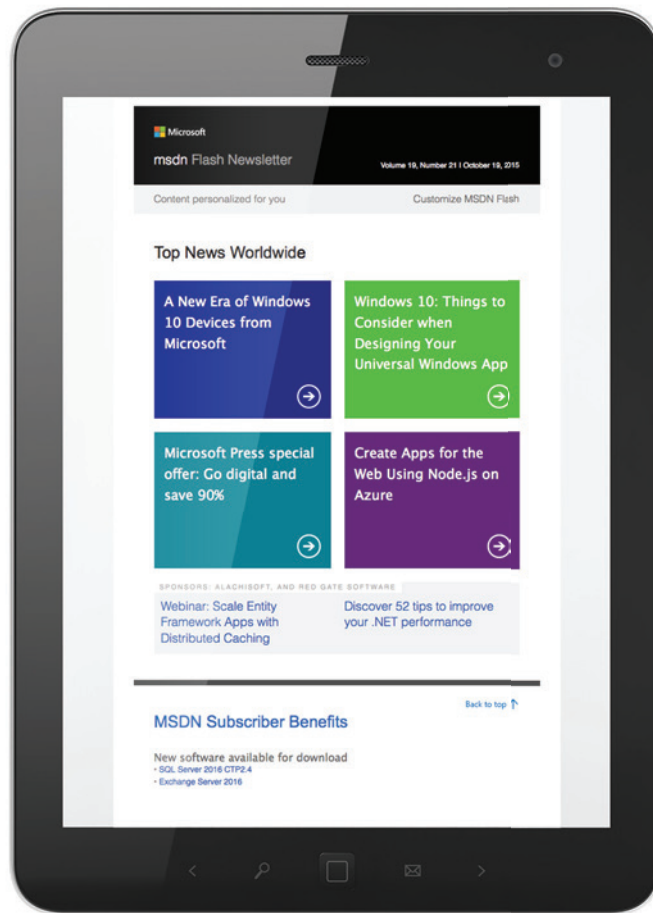
In my opinion, the best way to understand combinatorial evolution optimization is to start by examining the helper methods. Once the helpers are understood, the solve-method is relatively easy to grasp. Let me begin by explaining method RandomMatrix, which initializes the matrix field of an Organism object to a random possible solution. Somewhat surprisingly, method RandomMatrix is the trickiest part of the entire algorithm.

Figure 4 shows the definition of method RandomMatrix.

The algorithm is designed so that at any point in time, each of the nine 3x3 sub-grids is OK in the sense that each cell contains numbers 1 through 9, and there are no duplicate values.

The algorithm is designed so that at any point in time, each of the nine 3x3 sub-grids is OK in the sense that each cell contains numbers 1 through 9, and there are no duplicate values. A condition that's always true is sometimes called an invariant. This invariant influences all other methods of the algorithm. In theory, the row constraint or the column constraint could've been used as the invariant, but in practice using the sub-grid constraint is more effective.

Conceptually, walking through each 3x3 sub-grid in a 9x9 matrix isn't deep, but implementation is mildly tricky. The approach I took was to define two helper methods, Block and Corner. Method



Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn
magazine

msdn.microsoft.com/flashnewsletter

Figure 3 Demo Program Structure

```
using System;
using System.Collections.Generic;
namespace SudokuEvo
{
    class SudokuEvoProgram
    {
        static Random rnd = new Random(0);

        static void Main(string[] args)
        {
            Console.WriteLine("Begin solving Sudoku");
            Console.WriteLine("The problem is: ");
            int[][] problem = new int[9][];
            problem[0] = new int[] { 0, 0, 6, 2, 0, 0, 0, 8, 0 };
            problem[1] = new int[] { 0, 0, 8, 9, 7, 0, 0, 0, 0 };
            problem[2] = new int[] { 0, 0, 4, 8, 1, 0, 5, 0, 0 };
            problem[3] = new int[] { 0, 0, 0, 0, 6, 0, 0, 0, 2 };
            problem[4] = new int[] { 0, 7, 0, 0, 0, 0, 0, 3, 0 };
            problem[5] = new int[] { 6, 0, 0, 0, 5, 0, 0, 0, 0 };
            problem[6] = new int[] { 0, 0, 2, 0, 4, 7, 1, 0, 0 };
            problem[7] = new int[] { 0, 0, 3, 0, 2, 8, 4, 0, 0 };
            problem[8] = new int[] { 0, 5, 0, 0, 0, 1, 2, 0, 0 };

            DisplayMatrix(problem);

            int numOrganisms = 200;
            int maxEpochs = 5000;
            int maxRestarts = 20;

            int[][] soln = Solve(problem, numOrganisms,
                                maxEpochs, maxRestarts);
            Console.WriteLine("Best solution found: ");
            DisplayMatrix(soln);

            int err = Error(soln);
            if (err == 0)
                Console.WriteLine("Success \n");
            else
                Console.WriteLine("Did not find optimal solution \n");

            Console.WriteLine("End Sudoku demo");
            Console.ReadLine();
        } // Main()

        public static int[][] Solve(int[][] problem,
                                   int numOrganisms, int maxEpochs, int maxRestarts) { . . . }

        public static void DisplayMatrix(int[][] matrix) { . . . }

        public static int[][] SolveEvo(int[][] problem,
                                       int numOrganisms, int maxEpochs) { . . . }

        public static int[][] RandomMatrix(int[][] problem) { . . . }

        public static int[] Corner(int block) { . . . }

        public static int Block(int r, int c) { . . . }

        public static int[][] NeighborMatrix(int[][] problem,
                                             int[][] matrix)

        public static int[][] MergeMatrices(int[][] m1,
                                             int[][] m2) { . . . }

        public static int Error(int[][] matrix) { . . . }

        public static int[][] DuplicateMatrix(int[][] matrix) { . . . }

        public static int[][] CreateMatrix(int n) { . . . }
    } // Program

    public class Organism
    {
        public int type; // 0 = worker, 1 = explorer
        public int[][] matrix;
        public int error;
        public int age;

        public Organism(int type, int[][] m, int error, int age)
        {
            this.type = type;
            this.matrix = SudokuEvoProgram.DuplicateMatrix(m);
            this.error = error;
            this.age = age;
        }
    } // ns
```

Block accepts a row index *r* and a column index *c*, and returns a block number (0-8) that holds the cell at (*r*, *c*). Block numbers are assigned left-to-right, and then top-to-bottom. For example, if (*r*, *c*) = (3, 8) then method Block returns 5.

Method Corner accepts a block ID (0-8) and returns the indices of the upper-left-hand corner of the block. For example, if block = 8, method Corner returns (6, 6).

Once it's known that each of the nine 3x3 sub-grids of a matrix are OK, it's possible to define a relatively simple method that defines error:

```
public static int Error(int[][] matrix)
{
    int err = 0;
    for (int i = 0; i < 9; ++i) { // Each row
        // Determine missing values in row
        // Add 1 to err for each missing value
    }
    for (int j = 0; j < 9; ++j) { // each column
        // Determine missing values in column
        // Add 1 to err for each missing value
    }
    return err;
}
```

In words, the total error for a possible solution is the sum of the number of missing values in the rows, plus the number of missing values in the columns. Because of the algorithm invariant, all the 3x3 sub-grids have no missing values, so they don't contribute to error. Note that counting the number of duplicate values in each row and column is equivalent to counting the number of missing values.

Generating a Neighbor Matrix

In combinatorial evolution, there are two types of Organism objects. Those that are type explorer search for possible solutions randomly, using method RandomMatrix. Organism objects that are type worker repeatedly try to find a better solution to the one stored in their matrix field, by examining a close, "neighbor" possible solution.

Because of the 3x3 sub-grid invariant, a neighbor solution must confine itself to being a permutation of a sub-grid. Put more concretely, to determine a neighbor matrix, the algorithm selects a block at random, then selects two cells in the block (where neither cell contains a fixed value from the problem definition), and exchanges the values in the two cells.

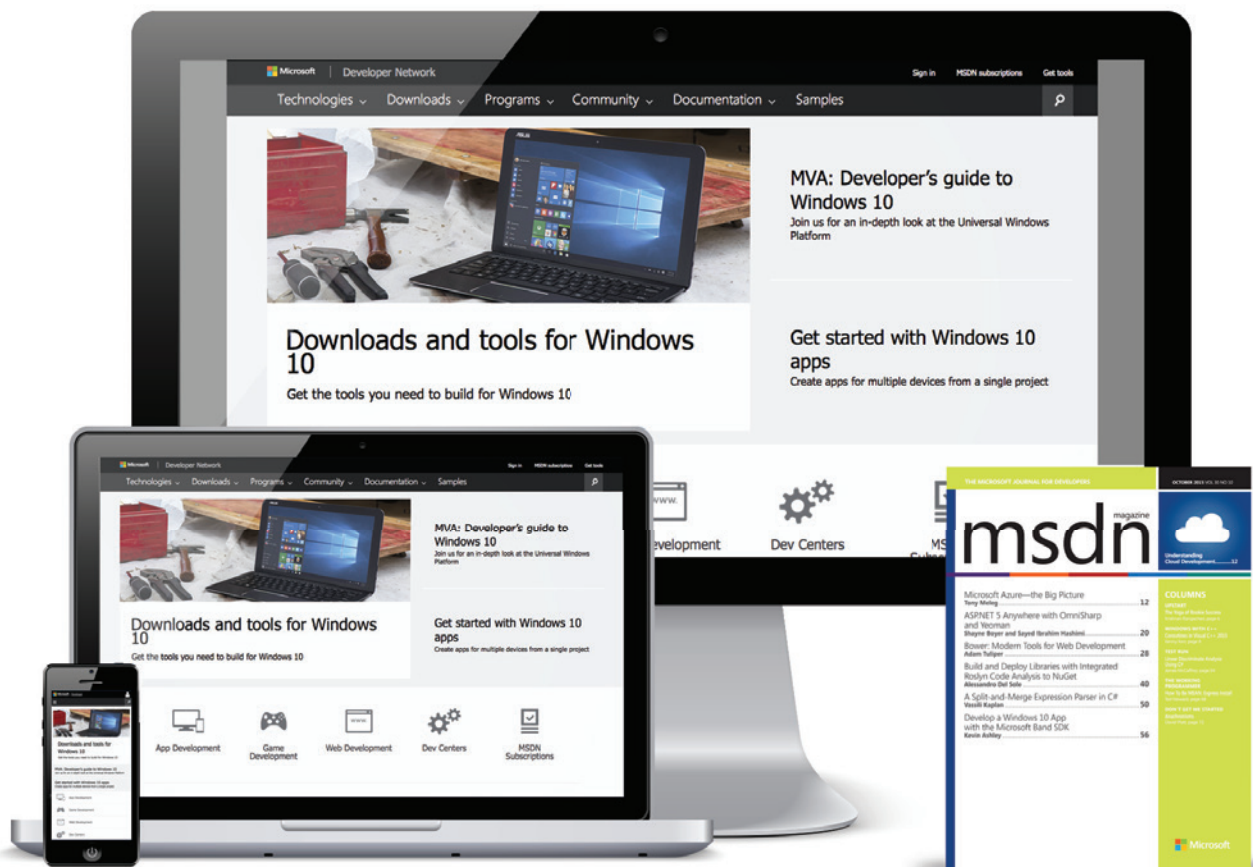
Figure 4 Definition of the RandomMatrix Method

```
public static int[][] RandomMatrix(int[][] problem)
{
    int[][] result = DuplicateMatrix(problem);
    for (int block = 0; block < 9; ++block) {
        // Create a List with values 1-9
        // Shuffle the values in List
        // Walk through each cell in current block
        // If a cell is occupied, remove that value from List
        // Walk through each cell in current block
        // If cell is empty, add value from List
    }
    return result;
}
```

msdn

magazine

Where you need us most.



Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!
360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

Method `NeighborMatrix` is defined like so:

```
public static int[][] NeighborMatrix(int[][] problem, int[][] matrix)
{
    int[][] result = DuplicateMatrix(matrix);
    int block = rnd.Next(0, 9); // pick a random block
    // Determine which cells have values that can be swapped
    // Pick two of those cells: (r1,c1) and (r2,c2)
    int tmp = result[r1][c1];
    result[r1][c1] = result[r2][c2];
    result[r2][c2] = tmp;
    return result;
}
```

The demo program assumes the existence of a class-scope `Random` object named `rnd`. This design is common in many optimization algorithms. The idea of generating a neighbor solution is found in several combinatorial optimization algorithms, for example, simulated annealing optimization and simulated bee colony optimization.

Merging Two Matrices

The combinatorial evolution optimization algorithm implements a form of virtual evolution by selecting two good `Organism` objects (meaning they have a matrix field with small error), and then using those good objects to create a new child object. The new, presumably very good, child `Organism` replaces a poor `Organism`.

Method `MergeMatrices` accepts two 9x9 matrices from two `Organism` objects. The method scans through blocks 0 to 8. For each block, a random value between 0.0 and 1.0 is generated. If the random value is less than 0.50 (that is, about half the time), then the values in the two blocks are exchanged. The code is:

```
public static int[][] MergeMatrices(int[][] m1, int[][] m2)
{
    int[][] result = DuplicateMatrix(m1);
    for (int block = 0; block < 9; ++block) {
        double pr = rnd.NextDouble();
        if (pr < 0.50) {
            // Replace values in block of m1 with those in m2
        }
    }
    return result;
}
```

This evolutionary mechanism is somewhat similar to the chromosome crossover mechanism used in genetic algorithms.

The SolveEvo Method

The primary algorithm is implemented in method `SolveEvo`. The method is best described with a combination of code and high-level pseudo-code, as shown in **Figure 5**.

Figure 5 Primary Algorithm Implemented in the `SolveEvo` Method

```
public static int[][] SolveEvo(int[][] problem,
    int numOrganisms, int maxEpochs)
{
    int numWorker = (int)(numOrganisms * 0.90);
    int numExplorer = numOrganisms - numWorker;
    Organism[] hive = new Organism[numOrganisms];
    // Initialize each Organism
    int epoch = 0;
    while (epoch < maxEpochs) {
        for (int i = 0; i < numOrganisms; ++i) {
            // Process each Organism
        }
        // Merge best worker with best explorer, increment epoch
    }
    return bestMatrix;
}
```

The method begins by determining the number of worker `Organism` objects as 90 percent of the total number used. This value was determined by trial and error. The `Organism` objects are stored in an array named `hive`.

The pseudo-code for processing a worker-type `Organism` is:

```
generate a neighbor matrix
if neighbor is better (or Organism makes mistake)
    update matrix with neighbor
    reset age to 0
else
    don't update matrix
    increment age
end-if
```

The algorithm occasionally (probability = 0.001) instructs an `Organism` object to accept a neighbor solution that's worse than the object's current solution. This is a common optimization strategy designed to help an algorithm get unstuck from a non-optimal solution.

At each epoch in the iteration loop, each explorer-type `Organism` generates a random solution grid. Then after each of the `Organism` objects has been processed, a new `Organism` is created. In pseudo-code:

```
determine index of best worker Organism
determine index of best explorer Organism
create new Organism using best worker and best explorer
determine index of worst worker Organism
replace worst worker with new child Organism
```

As it turns out, this evolutionary mechanism is critical to the algorithm's success. Without evolution, the algorithm sometimes fails, but with evolution, the algorithm has successfully solved every difficult Sudoku problem I've presented it with, including some heinously difficult problems I found on the Internet. However, combinatorial optimization hasn't been subjected to research analysis, so the fact that combinatorial evolution can solve Sudoku puzzles is no guarantee that it can solve arbitrary combinatorial optimization problems.

Wrapping Up

The combinatorial evolution algorithm I've presented in this article isn't really an algorithm. Combinatorial evolution is a meta-heuristic. By that I mean combinatorial evolution is just a set of general guidelines that can be used to design a concrete algorithm to solve a specific optimization problem.

It's unlikely you'll need to solve a Sudoku problem in your regular work environment, but combinatorial optimization can be used to solve real-life problems, too. The key ideas in combinatorial optimization are to define a data structure that describes the target problem; define what is meant by a random solution; define what a neighbor solution is; and define an error metric. With these pieces of the puzzle in place, many problems that can't be solved by traditional algorithms can be solved quickly and efficiently using combinatorial evolution. ■

DR. JAMES McCaffrey works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products including Internet Explorer and Bing. Dr. McCaffrey can be reached at jammc@microsoft.com.

THANKS to the following Microsoft technical experts who reviewed this article: Chris Lee and Kirk Olynyk



ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**DEC
5-9**

*** REGISTER WITH DISCOUNT
CODE L360NOV AND
SAVE \$300!**

SQL Server® **LIVE!**

TRAINING FOR DBAs AND IT PROS

Lead the Data Race

After 5 days of workshops, deep dives and breakout sessions, SQL Server Live! will leave you with the skills needed to Lead the Data Race.

With timely, relevant content, SQL Server Live! helps administrators, DBAs, and developers do more with their SQL Server investment. Sessions will cover performance tuning, security, reporting, data integration, adopting new techniques, improving old approaches, and modernizing the SQL Server infrastructure.



Must use discount code L360NOV

Scan the QR code to register or for more event details.



A Part of Live! 360: The Ultimate Education Destination

6 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio **LIVE!**

ModernApps **LIVE!**

TECHMENTOR

Office & SharePoint **LIVE!**

SQL Server **LIVE!**

APPDEV TRENDS NEW!

SQLLIVE360.COM

EVENT PARTNERS



PLATINUM SPONSORS



GOLD SPONSORS



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY





How To Be MEAN: Take a Gulp

Welcome back, MEANers.

If you read my last column in October, which talked about the “reboot” of the codebase (by using Yeoman to scaffold out the basics and glue code), you might have noticed that a new tool was thrown into the mix without much explanation. I refer, of course, to the “Gulp” tool used to start the server and open the client browser to the scaffolded client application (msdn.com/magazine/mt742874).

If you missed my last column, it’s easy enough to catch up. First, make sure Yeoman and the “angular-fullstack” generators are both installed on your Node.js development environment (along with a local running copy of MongoDB):

```
npm install -g yeoman angular-fullstack-generator
yo angular-fullstack
```

Then, after answering questions as to what tools Yeoman should scaffold into place (for this column, the choices made are mostly irrelevant), and after Yeoman has helpfully kicked off an “npm install” to pull down all the runtime and development dependencies, the scaffolder will report that the application is ready to test by running “gulp test” or “gulp start:server.”

Clearly, whatever Gulp is, it’s some kind of build tool, akin in spirit to Make, MSBuild or Ant. But how it works is a little different than any of those three tools, and it deserves discussion as a result.

Just a Little Gulp to Start

While it’s not exactly fair to call Gulp a “build tool” for a language that doesn’t ever get built (remember, ECMAScript is generally intended to be an interpreted language), that’s really the best terminology you have for a tool that’s intended to be run after (or during) development to make sure everything is lined up and ready to go. Perhaps a better name for all of them would be “development automation tool,” because that’s more accurate and would include the act of compiling code and assembling it into a deployable artifact.

Figure 1 Continuous Automation Using Gulp

```
// Don't need it in this file, but you need it installed
require('jshint');

const gulp = require('gulp');
const jshint = require('gulp-jshint');

gulp.task('default', ['watch']);

gulp.task('watch', function() {
  gulp.watch(['*.js', '!gulpfile.js', 'client/**/*.js', 'server/**/*.js'],
    ['jshint']);
});

gulp.task('jshint', function() {
  return gulp.src(['*.js', '!gulpfile.js', 'client/**/*.js', 'server/**/*.js'])
    .pipe(jshint())
    .pipe(jshint.reporter('default'));
});
```

However, because that’s a mouthful and “build tool” just rather rolls off the tongue, let’s go with the idea that Gulp is a build tool for now.

To get started with Gulp, let’s break away from the scaffolded code from earlier and start from scratch to focus on what (and how) Gulp does what it does. Install the global Gulp command-line tools (“npm install --g gulp-cli”) first. Then, in a new directory, create an empty Node.js project by running the following:

```
npm init
npm install --save-dev gulp
```

This will make sure that Gulp is referenced in the package.json file as a developer dependency, so that if you pull down the project you won’t have to remember to install Gulp—it’ll just come along for the ride with the next “npm install” in a fresh environment.

Now, in your text editor of choice, create a new file, called gulpfile.js:

```
const gulp = require('gulp');

gulp.task('default', function() {
  console.log("Gulp is running!");
});
```

Then, from the same directory, issue the stock Gulp command, which (perhaps unsurprisingly) is just “gulp.” Gulp thinks about it for a second, then comes back with:

```
[18:09:38] Using gulpfile ~/Projects/code/gulpdemo/gulpfile.js
[18:09:38] Starting 'default'...
Gulp is running!
[18:09:38] Finished 'default' after 142 μs
```

Not bad. Seems a little overkill at the moment, but not bad.

Gulp Down Some Tasks

Gulp, like most build tools, thinks in terms of “tasks,” and more important, how to track the dependencies between those tasks. So, in the simple Gulpfile here, it sees that there’s one task, called “default” (which is the well-understood convention to mean the task that should be executed if none is specified on the command line), and executes the body of the associated function literal when asked to execute that task. Naming the task differently is trivial:

```
const gulp = require('gulp');

gulp.task('echo', function() {
  console.log("Gulp is running!");
});

const gulp = require('gulp');

gulp.task('echo', function() {
  console.log("Gulp is running on " + (new Date()));
});
```

It should be apparent that Gulp is just code, so anything that can be done in code can also be done in the body of a Gulp task. This offers up many incredible options, like reading from a database to find elements that need to be code-generated, talking to other online services for configuration, or even just printing out the current date and time:



ORLANDO
ROYAL PACIFIC RESORT AT
UNIVERSAL ORLANDO

**DEC
5-9**

*** REGISTER WITH DISCOUNT
CODE L360NOV AND
SAVE \$300!**

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Taking Collaboration on the Road

Today, organizations expect people to work from anywhere at any time. Office & SharePoint Live! provides leading-edge knowledge and training to administrators, developers, and planners who must customize, deploy and maintain SharePoint Server on-premises and in Office 365 to maximize the business value.

Whether you are a Manager, IT Pro, DBA, or Developer, Office & SharePoint Live! brings together the best the industry has to offer for 5 days of workshops, keynotes, and sessions to help you work through your most pressing collaboration projects.



Must use discount code L360NOV

Scan the QR code to register or for more event details.



A Part of Live! 360: The Ultimate Education Destination

6 GREAT CONFERENCES, 1 GREAT PRICE

Visual Studio **LIVE!**

ModernApps **LIVE!**

TECHMENTOR

Office & SharePoint **LIVE!**

SQL Server **LIVE!**

APPDEV TRENDS NEW!

SPLIVE360.COM

EVENT PARTNERS



PLATINUM SPONSORS



GOLD SPONSORS



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



This produces the following:

```
Teds-MacBook-Pro:gulpdemo tedneward$ gulp echo
[18:16:24] Using gulpfile ~/Projects/code/gulpdemo/gulpfile.js
[18:16:24] Starting 'echo'...
Gulp is running on Wed Sep 14 2016 18:16:24 GMT-0700 (PDT)
[18:16:24] Finished 'echo' after 227 μs
```

Dependent tasks are simply listed as a string array in between the name of the task and its function literal body, so having task “echo” depending on another task simply looks like this:

```
const gulp = require('gulp');
const child_process = require('child_process');

gulp.task('gen-date', function() {
  child_process.exec('sh date > curdate.txt');
});

gulp.task('echo', ['clean', 'gen-date'], function() {
  console.log("Gulp is running on " + (new Date()));
});
```

Here, the “gen-date” task makes use of a standard Node.js package, “child_process,” to kick off an external tool to write the date to a file, just to prove that you can. Which, honestly, is all well and good, but generally build tools are expected to do something of greater import than just write stuff to the console and figure out the current date and time.

Gulping a Little More

Let's put a bit more meat into this. Create an index.js file with the following ECMAScript code in it, including the not-quite-good code parts:

```
// index.js
function main(args) {
  for (let arg in args) {
    if (arg === "hello")
      console.log("world!");
    console.log("from index.js!");
  }
}

console.log("Hello, from index.js!")
main()
```

Yes, it's a bit nonsensical and, yes, it clearly has a few issues with it, but that's the whole point—it would be nice if there were a tool that could spot some of those issues and report on them. (Dare I say, “Compile-time check”?) Fortunately, such a tool exists in JSHint (jshint.com), but it installs by default as a command-line tool and it would be a pain to have to remember to run it all the time.

Fortunately, this is what a build tool is for. Going back to the Gulpfile, let's get Gulp to run JSHint over each of the source files (of which there is only one right now) by telling it about the source files in question and then asking it to execute JSHint over each:

```
// Don't need it in this file, but we need it installed
require('jshint');
require('jshint-stylish');

const gulp = require('gulp');
const jshint = require('gulp-jshint');

gulp.task('jshint', function() {
  return gulp.src('*.js')
    .pipe(jshint())
    .pipe(jshint.reporter('jshint-stylish'));
});
```

When you run this, the tool will point out that there are some suggested changes to all of the “js” files in the current directory, including the Gulpfile itself. Bah. You don't really care about checking the Gulpfile, so let's screen it out of the collection of files to run:

```
gulp.task('jshint', function() {
  return gulp.src(['*.js', '!gulpfile.js'])
    .pipe(jshint())
    .pipe(jshint.reporter('jshint-stylish'));
});
```

This will remove “gulpfile.js” from the current list of files before passing it on to the next stage in the pipeline. As a matter of fact, you'll probably want most code to live in a “src” directory (or “server” and “client” directories, for each side), so add those and any of their subdirectories to the list of files to process:

```
gulp.task('jshint', function() {
  return gulp.src(['*.js', '!gulpfile.js', 'server/**/*.js', 'client/**/*.js'])
    .pipe(jshint())
    .pipe(jshint.reporter('jshint-stylish'));
});
```

The “double-star” in each path gets the recursive behavior to pick up any “js” files in each of those subdirectories.

On the surface of it, this is great, but it doesn't really do much for you: You still have to type “gulp jshint” (or just “gulp” if you tied it into the “default” task as a dependency) by hand each time you want to see what needs fixing. Why can't you just have the thing run any time the code changes, the way IDEs do?

Sure, do that, as shown in **Figure 1**.

Now, when you run “gulp,” the command line will simply pause and wait. Gulp is now in “watch” mode, wherein it'll keep an eye on any of the files passed to the “gulp.watch” call, and if any of those files changes (meaning, they're saved—Gulp can't peer inside of text editors, unfortunately), it'll immediately run the “jshint” task on the complete set of files. And keep watching.

Gulping More

One of the keys to understanding how Gulp tasks process files is buried in the call to pipe. Gulp thinks in terms of “streams,” rather than tasks or files. For example, a simple Gulp task that would copy files from the “src” directory to the “dest” directory would look like this:

```
gulp.task('copy-files', function() {
  gulp.src('source/folder/**')
    .pipe(gulp.dest('dest/folder/**'));
});
```

In essence, the files are each picked up by gulp.src, and deposited, untouched, into the destination specified by gulp.dest. Anything that needs to happen to those files simply goes in as a step in the pipeline, and each file flows through that pipeline before proceeding to the next step in the pipeline. It's the Unix architectural style of “pipes and filters,” brought to the Node.js ecosystem in an incredibly elegant way. Windows PowerShell is built on the same kind of architecture, for those who are thinking they've seen something like this before in the .NET universe.

Therefore, for example, if you just want to see Gulp touch each file through the pipeline, there's a plug-in for that (“gulp-filelogger”), and it'll print to the console on each file it touches:

```
gulp.task('copy-files', function () {
  gulp.src(srcFiles)
    .pipe(filelogger())
    .pipe(gulp.dest(destDir));
});
```

This results in the following:

```
Teds-MacBook-Pro:gulpdemo tedneward$ gulp copy-files
[20:14:01] Using gulpfile ~/Projects/code/gulpdemo/gulpfile.js
[20:14:01] Starting 'copy-files'...
[20:14:01] Finished 'copy-files' after 14 ms
[20:14:01] [/Users/tedneward/Projects/code/gulpdemo/index.js]
Teds-MacBook-Pro:gulpdemo tedneward$
```

Figure 2 A Gulp Recipe

```
var gulp = require('gulp');
var runSequence = require('run-sequence');
var conventionalChangelog = require('gulp-conventional-changelog');
var conventionalGithubReleaser = require('conventional-github-releaser');
var bump = require('gulp-bump');
var gutil = require('gulp-util');
var git = require('gulp-git');
var fs = require('fs');

gulp.task('changelog', function () {
  return gulp.src('CHANGELOG.md', {
    buffer: false
  })
  .pipe(conventionalChangelog({
    preset: 'angular' // Or to any other commit message convention you use.
  }))
  .pipe(gulp.dest('./'));
});

gulp.task('github-release', function(done) {
  conventionalGithubReleaser({
    type: "oauth",
    token: '' // Change this to your own GitHub token.
  }, {
    preset: 'angular' // Or to any other commit message convention you use.
  }, done);
});

gulp.task('bump-version', function () {
  // Hardcode the version change type to "patch," but it might be a good
  // idea to use minimist (bit.ly/2cyPhfa) to determine with a command
  // argument whether you're doing a "major," "minor" or a "patch" change.
  return gulp.src(['./bower.json', './package.json'])
  .pipe(bump({type: "patch"}).on('error', gutil.log))
  .pipe(gulp.dest('./'));
});

gulp.task('commit-changes', function () {
  return gulp.src('.')
  .pipe(git.add())
  .pipe(git.commit('[Prerelease] Bumped version number'));
});

gulp.task('push-changes', function (cb) {
  git.push('origin', 'master', cb);
});

gulp.task('create-new-tag', function (cb) {
  var version = getPackageJsonVersion();
  git.tag(version, 'Created Tag for version: ' + version, function (error) {
    if (error) {
      return cb(error);
    }
    git.push('origin', 'master', {args: '--tags'}, cb);
  });
});

function getPackageJsonVersion () {
  // Parse the json file instead of using require because require caches
  // multiple calls so the version number won't be updated.
  return JSON.parse(fs.readFileSync('./package.json', 'utf8')).version;
};

gulp.task('release', function (callback) {
  runSequence(
    'bump-version',
    'changelog',
    'commit-changes',
    'push-changes',
    'create-new-tag',
    'github-release',
    function (error) {
      if (error) {
        console.log(error.message);
      } else {
        console.log('RELEASE FINISHED SUCCESSFULLY');
      }
      callback(error);
    }
  );
});
```

Notice how the output appears after Gulp reports it's finished. Gulp can (and does) process these streams asynchronously much of the time, shortening the "build" times. Most of the time, developers neither know nor care that things are being done in parallel, but for those times that performing things in a precise sequence is important, not surprisingly, the Gulp plug-in community has some plug-ins that will serialize execution and make sure everything happens in sequence. Gulp 4.0 will add two new functions, `parallel` and `serial`, to make this more clear, but because it hasn't shipped yet, you'll have to wait on those.

By the way, Gulp itself consists purely of the four functions you've seen so far: `gulp.task`, `gulp.watch`, `gulp.src` and `gulp.dest`. Everything else is all plug-ins, npm modules or written by hand. This makes Gulp in and of itself extremely easy to understand. Depressingly easy, in fact, for article authors who are paid by the word.

Gulping a Lot All at Once

Gulp on its own is not all that complicated a tool, but as with any tool of this nature, its real strength lies in the vast array of plug-ins and complementary tools that have emerged out of the community around it. The full list is available at gulpjs.com/plugins, but **Figure 2** shows a representative sample of a Gulp recipe, demonstrating how to automate the release of a project to GitHub, including the Git commands to push to master.

This example demonstrates a number of things: how to run tasks in a particular sequence, using the Gulp plug-ins for generating a convention change set file, doing GitHub-style release messages, bumping the semantic version and a whole lot more. All from `gulp` release; that's pretty powerful.

Wrapping Up

This hasn't been a particularly code-heavy article, yet you just rebooted the entire application, gained a whole lot of functionality, and essentially brought the application up to the same level (and beyond) from what you'd been building for the past year or so. Gotta love scaffolding!

More important, having built all the parts piece by piece by hand prior to running the scaffolding, it's much easier to understand the code as a whole and what's happening where. For example, opening up `routes.js` will look familiar to the routing table you built by hand earlier, and the `package.json` (in the root of the project directory) will be bigger, but will remain the same as you'd been using.

The only new thing, in fact, beyond the use of Yeoman itself, is the introduction of a build tool to gather all the pertinent parts together into the right place, and that'll be what I discuss next time. Until then ... happy coding! ■

TED NEWARD is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP, has authored and coauthored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.

THANKS to the following technical expert for reviewing this article:
Shawn Wildermuth

ROCK YOUR CODE TOUR 2017

LAS VEGAS



**MARCH
13-17**



AUSTIN



**MAY
15-18**



WASH DC



**JUNE
12-15**



SUPPORTED BY



Visual Studio
MAGAZINE

PRODUCED BY

1105MEDIA⁷
YOUR GROWTH. OUR BUSINESS.

REDMOND



AUG
14-18



CHICAGO



SEPT
18-21



ANAHEIM



OCT
16-19



ORLANDO



NOV
13-17



CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search “VSLive”



linkedin.com – Join the
“Visual Studio Live” group!

vslive.com



Add Facial Recognition Features to Your App

At Build 2016, Microsoft announced the release of the Cognitive Services API. Among the many APIs available are several computer vision services. These services can analyze the age and gender of faces in an input image. There's even an API for detecting individuals' emotions based on their facial expressions. To highlight the technology, there were numerous kiosks throughout the event space demonstrating various uses of the technology. The Cognitive Services API leverages Microsoft's experience and efforts in the space of machine learning. Thousands of labeled images were fed through a neural network. Best of all, you can leverage these services without any knowledge of machine learning or artificial intelligence. You simply call a Web service from your app. You can watch an interview with one of the team members involved with the project to learn more about the process at bit.ly/1TG1QK.

With Cognitive Services APIs, you can add basic facial detection to your app without calling any APIs. The `Windows.Media.FaceAnalysis` namespace contains functionality to detect faces in images or videos. The feature set is basic and lacks the rich data set of Cognitive Services. In fact, it's very similar to facial detection found in many digital cameras. While the features are basic, they have two distinct advantages: They work offline and, because you're not calling an API, you won't incur any charges. As an optimization strategy, your app can detect the presence of a face locally before calling the Cognitive Services API. This way the app won't send images without faces to the Cognitive Services API. That can amount to a significant cost savings for you and reduced bandwidth usage for your users. Detecting faces locally can be a useful augmentation to intelligent cloud services like Cognitive Services.

Setting Up the Project

In Visual Studio 2015, create a new Universal Windows Platform (UWP) app project, choose the Blank template, and name it `FaceDetection`. Because the app will use the webcam, you must

add that capability to the app. In Solution Explorer, double-click on the `Package.appxmanifest` file. In the Capabilities tab, check the checkboxes next to Microphone and Webcam, as shown in **Figure 1**. Save the file.

Now, add the following XAML to the `MainPage.xaml` file to create the UI:

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="320*" />
    <RowDefinition Height="389*" />
  </Grid.RowDefinitions>

  <CaptureElement Name="cePreview" Stretch="Uniform" Grid.Row="0" />
  <Canvas x:Name="cvsFaceOverlay" Grid.Row="0" />

  <StackPanel Grid.Row="1" HorizontalAlignment="Center" Margin="5">
    <Button x:Name="btnCamera" Click="btnCamera_Click" >Turn on Camera</Button>
    <Button x:Name="btnDetectFaces" Click="btnDetectFaces_Click" >Detect Faces</Button>
  </StackPanel>
</Grid>
```

You might not be familiar with the `CaptureElement` control. The `CaptureElement` control renders a stream from an attached capture

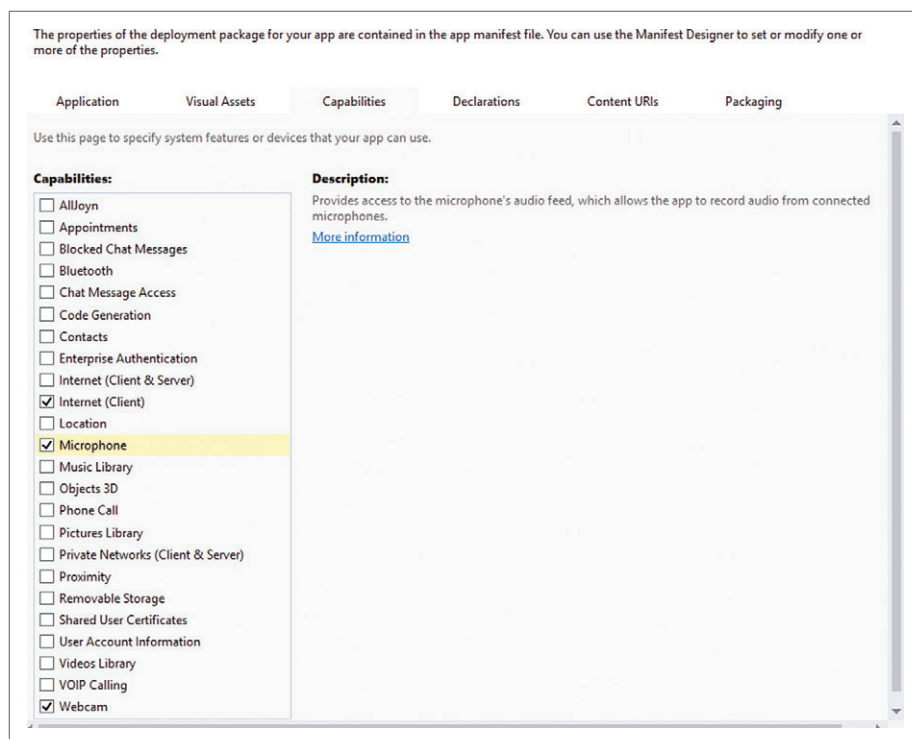


Figure 1 Adding Webcam and Microphone Capabilities to the App

Code download available at bit.ly/2bez0XA.

Figure 2 Adding a Rectangle Object to the Faces Overlay Canvas

```
private void DrawFaceBoxes(IReadOnlyList<DetectedFace> detectedFaces)
{
    cvsFaceOverlay.Children.Clear();

    for (int i = 0; i < detectedFaces.Count; i++)
    {
        var face = detectedFaces[i];
        var faceBounds = face.FaceBox;

        Rectangle faceHighlightRectangle = new Rectangle()
        {
            Height = faceBounds.Height,
            Width = faceBounds.Width
        };

        Canvas.SetLeft(faceHighlightRectangle, faceBounds.X);
        Canvas.SetTop(faceHighlightRectangle, faceBounds.Y);

        faceHighlightRectangle.StrokeThickness = 2;

        faceHighlightRectangle.Stroke = new SolidColorBrush(Colors.Red);
        cvsFaceOverlay.Children.Add(faceHighlightRectangle);
    }
}
```

device, usually a camera or webcam. In the codebehind, you'll use the MediaCapture API to connect it to a stream from the webcam.

Previewing the Video from the Camera

In the MainPage.xaml.cs file add the following namespaces:

```
using Windows.Media.Capture;
using Windows.Media.Core;
using Windows.Media.FaceAnalysis;
```

Next, add the two following members to the MainPage class:

```
private FaceDetectionEffect _faceDetectionEffect;
private MediaCapture _mediaCapture;
private IMediaEncodingProperties _previewProperties;
```

Now, add the following event handler for the Start Camera button:

```
private async void btnCamera_Click(object sender, RoutedEventArgs e)
{
    _mediaCapture = new MediaCapture();
    await _mediaCapture.InitializeAsync();

    cePreview.Source = _mediaCapture;
    await _mediaCapture.StartPreviewAsync();
}
```

Each detected face has a bounding box of where on the image the face was detected.

Run the project and then click the Start Camera button. You should now see the output of your webcam in the app. If you don't have a webcam attached to your system, then an exception will be thrown.

Tracking Faces

With the CaptureElement control successfully streaming video from the Webcam, now it's time to start tracking faces. Tracking faces requires the creation of a FaceDetectionDefinition object, setting some properties on the object, and then connecting it to the _mediaCapture object that streams the video to the CaptureElement created.

Inside the event handlers for the Detect Faces button, add the following code:

```
private async void btnDetectFaces_Click(object sender, RoutedEventArgs e)
{
    var faceDetectionDefinition = new FaceDetectionEffectDefinition();
    faceDetectionDefinition.DetectionMode = FaceDetectionMode.HighPerformance;
    faceDetectionDefinition.SynchronousDetectionEnabled = false;
    _faceDetectionEffect = (FaceDetectionEffect) await
        _mediaCapture.AddVideoEffectAsync(faceDetectionDefinition,
            MediaStreamType.VideoPreview);
    _faceDetectionEffect.FaceDetected += FaceDetectionEffect_FaceDetected;
    _faceDetectionEffect.DesiredDetectionInterval = TimeSpan.FromMilliseconds(33);
    _faceDetectionEffect.Enabled = true;
}
```

This code creates a FaceDetectionDefinition object that's optimized for performance. This can be seen in the line of code where DetectionMode is set to HighPerformance. The FaceDetectionMode enumeration has three members: HighPerformance prioritizes speed over accuracy, HighQuality prioritizes accuracy over speed, and Balanced finds a compromise between accuracy and speed. The next line of code doesn't delay incoming video frames while running the face detection algorithms. This keeps the preview video running smoothly.

Next, the FaceDetectionDefinition is added to the MediaCapture object, along with an enumeration specifying the type of media in the stream. Once added, a FaceDetectionEffect object is returned. This object has a FaceDetected event that fires when a face is detected, a DesiredDetectionInterval property that sets the frequency of face detection, and an Enabled property that enables or disables face detection.

Drawing Rectangles Around Faces

Now that the FaceDetectionEffect has been added to the MediaCapture object and enabled, it's time to add code to the FaceDetection event handler:

```
private async void FaceDetectionEffect_FaceDetected(
    FaceDetectionEffect sender, FaceDetectedEventArgs args)
{
    var detectedFaces = args.ResultFrame.DetectedFaces;
    await Dispatcher
        .RunAsync(CoreDispatcherPriority.Normal, () => DrawFaceBoxes(detectedFaces));
}
```

As this event runs on another thread, you must use the Dispatcher to make changes to the UI thread. The next line of code iterates

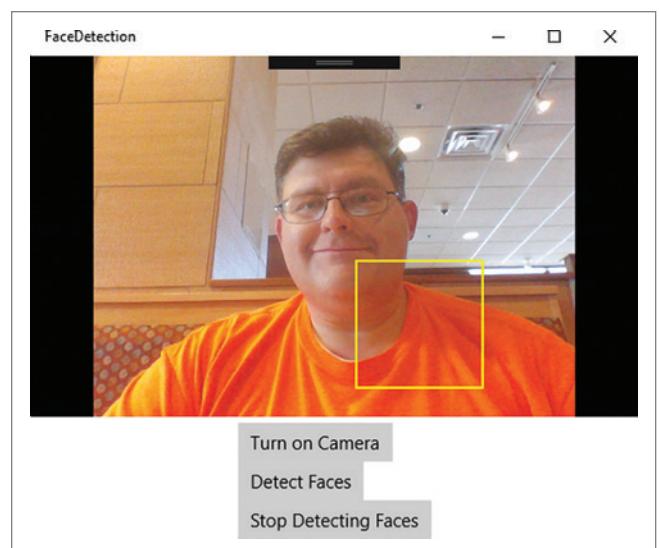


Figure 3 Face Detected but Location in the UI Isn't Right

through the `IReadOnlyList` of detected faces. Each detected face has a bounding box of where on the image the face was detected. Based on that data, you then create a new `Rectangle` object and add that to the faces overlay canvas, as shown in **Figure 2**.

Run the solution now and you'll notice something is "off" about the rectangles, as shown in **Figure 3**.

Finding the Correct Offset

The reason that rectangles are off is that the face detection algorithm's pixel grid starts at the top left of the media stream and not the representation of it shown in the UI. You must also consider that the resolution of the video feed from the camera may differ from the resolution in the UI. To place the rectangle in the proper

place, you must take both the position and scale differences in the UI and the video stream into account. To accomplish this, you'll add two functions that'll do the work: `MapRectangleToDetectedFace` and `LocatePreviewStreamCoordinates`.

The first step is to retrieve information about the preview stream. You do this by casting the class wide `_previewProperties` to a `VideoEncodingProperties` object. `VideoEncodingProperties` describes the format of a video stream. Primarily, you want to know the stream's height and width. With that information, you can determine the aspect ratio of the media stream and whether it's different than the `CaptureElement` control.

The `LocatePreviewStreamCoordinates` method compares the media stream height and width to those of the `CaptureElement` control. Depending on the aspect ratio differences between the two, one of three cases are possible: The aspect ratios are the same and there will be no adjustment. If the aspect ratios differ, then letterboxes will be added. If the `CaptureElement`'s aspect ratio is greater than the media stream's aspect ratio, then letterboxes are added to the sides.

If letterboxes are added to the sides, then the face rectangle's X coordinate must be adjusted. If the media stream's aspect ratio is greater than the `CaptureElement`, then letterboxes are added above and below the video. In that case, then the face rectangle's Y coordinate must be adjusted.

With the placement of the letterboxes taken into account, you now must determine the difference in scaling between the media stream and the `CaptureElement` control in the UI. With a `Rectangle`, there are four elements to set: top, left, width and height. Top and left are dependency properties. For a good overview of `Dependency Properties`, read the article at bit.ly/2bqvsVY.

Run the solution again, as shown in **Figure 4**, and you should see the placement of the face highlight rectangle to be more accurate, as seen in **Figure 5**.

Stopping Face Detection

Face detection consumes processing power and, in battery-powered mobile devices, can lead to significantly reduced battery life. Once you have face detection on, you might wish to give users the option to turn it off.

Figure 4 Code to Calculate the Correct Offset

```
private Rectangle MapRectangleToDetectedFace(BitmapBounds detectedfaceBoxCoordinates)
{
    var faceRectangle = new Rectangle();
    var previewStreamProperties =
        _previewProperties as VideoEncodingProperties;

    double mediaStreamWidth = previewStreamProperties.Width;
    double mediaStreamHeight = previewStreamProperties.Height;

    var faceHighlightRect = LocatePreviewStreamCoordinates(previewStreamProperties,
        this.cePreview);

    faceRectangle.Width = (detectedfaceBoxCoordinates.Width / mediaStreamWidth) *
        faceHighlightRect.Width;
    faceRectangle.Height = (detectedfaceBoxCoordinates.Height / mediaStreamHeight) *
        faceHighlightRect.Height;

    var x = (detectedfaceBoxCoordinates.X / mediaStreamWidth) *
        faceHighlightRect.Width;
    var y = (detectedfaceBoxCoordinates.Y / mediaStreamHeight) *
        faceHighlightRect.Height;

    Canvas.SetLeft(faceRectangle, x);
    Canvas.SetTop(faceRectangle, y);

    return faceRectangle;
}

public Rect LocatePreviewStreamCoordinates(
    VideoEncodingProperties previewResolution,
    CaptureElement previewControl)
{
    var uiRectangle = new Rect();

    var mediaStreamWidth = previewResolution.Width;
    var mediaStreamHeight = previewResolution.Height;

    uiRectangle.Width = previewControl.ActualWidth;
    uiRectangle.Height = previewControl.ActualHeight;

    var uiRatio = previewControl.ActualWidth / previewControl.ActualHeight;
    var mediaStreamRatio = mediaStreamWidth / mediaStreamHeight;

    if (uiRatio > mediaStreamRatio)
    {
        var scaleFactor = previewControl.ActualHeight / mediaStreamHeight;
        var scaledWidth = mediaStreamWidth * scaleFactor;

        uiRectangle.X = (previewControl.ActualWidth - scaledWidth) / 2.0;
        uiRectangle.Width = scaledWidth;
    }
    else
    {
        var scaleFactor = previewControl.ActualWidth / mediaStreamWidth;
        var scaledHeight = mediaStreamHeight * scaleFactor;
        uiRectangle.Y = (previewControl.ActualHeight - scaledHeight) / 2.0;
        uiRectangle.Height = scaledHeight;
    }

    return uiRectangle;
}
```

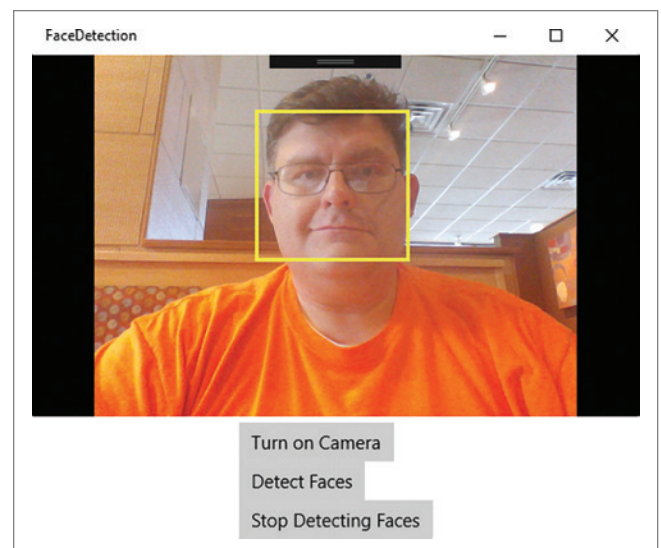


Figure 5 More Accurate Placement of Face Highlight Rectangle

Fortunately, turning face detection off is fairly straightforward. First, add a button to the StackPanel in the MainPage.xaml file:

```
<Button x:Name="btnStopDetection" Click="btnStopDetection_Click">Stop  
Detecting Faces</Button>
```

Now, add the following code to the event handler for the Stop Detecting Faces button:

```
private async void btnStopDetection_Click(object sender, RoutedEventArgs e)  
{  
    _faceDetectionEffect.Enabled = false;  
    _faceDetectionEffect.FaceDetected -= FaceDetectionEffect_FaceDetected;  
    await _mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);  
    _faceDetectionEffect = null;  
}
```

The code essentially undoes the setup process. The face detection effect is disabled, the FaceDetected event is unsubscribed from the event handler and the effect is cleared from the media capture object. Finally, the _faceDetectionEffect is set to null to free up memory.

Run the project now. Click on the Start Camera button, then Detect Faces and, last, Stop Detecting Faces. You should notice that even though the app is no longer detecting faces, there's still a rectangle in the last location a face was detected. Let's fix that.

Stop the app and go back to the btnStopDetection_Click event handler and add the following line of code to clear the contents of the cvsFacesOverlay canvas:

```
this.cvsFaceOverlay.Children.Clear();
```

Run the solution again and repeat all the steps. Now, when face detection is turned off, there are no rectangle highlights.

Wrapping Up

The Cognitive Services APIs provide easy access to powerful computer vision algorithms. Like all cloud services, however, they require Internet access to work. For some use cases that require offline scenarios, you can still perform basic facial detection by using the Face Detection APIs built right into the UWP.

Offline use cases can include placing a webcam attached to a Raspberry Pi 2 running Windows IoT Core in a remote location. The app would then save images locally when it detected a face. When data from the device was collected, the images could then be uploaded to Cognitive Services for advanced analysis.

Performing face detection locally also optimizes the bandwidth transmission and cloud service usage by allowing developers to only upload images with faces in them. In short, local Face Detection can augment online scenarios and empower offline uses, as well.

For a deeper dive, be sure to look at the CameraFaceDetection sample in the UWP sample apps on GitHub (bit.ly/2b27g1k). ■

FRANK LA VIGNE is a technology evangelist on the Microsoft Technology and Civic Engagement team, where he helps users leverage technology to create a better community. He blogs regularly at FranksWorld.com and has a YouTube channel called *Frank's World TV* (youtube.com/FranksWorldTV).

THANKS to the following technical expert for reviewing this article:
Rachel Appel

#ROWSHARE

Your free online **collaborative table**

Keep organized with RowShare, a simple, timesaving tool for gathering and sharing information.

1

INTUITIVE
TABLE DESIGN

2

SMART
ROWS AND ROW SHARING

3

SIMPLE
DOCUMENT CREATION

	Last Name	First Name	Position	Vote	Photo	Reminder	CV
	J. Brown	Thomas	Marketing	2		6/23/2016 10:30 AM	CV_Thomas_JBrown.pdf
	Speerman	Linda	Sales	4		6/28/2016 2:00 PM	CV_Linda_Speerman.pdf
	Arinez	Nathan	Sales	3		8/3/2016 5:30 PM	CV_Nathan_Martinez.pdf
	Flowers	Angela Heloise Cardigagno	Consultant	2		7/28/2016 2:00 PM	CV_Angela_Flowers.pdf

www.rowshare.com

Visual Studio[®] LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MAR 13-17 2017
BALLY'S, LAS VEGAS, NV



EVENT PARTNERS



Magenic

SUPPORTED BY



msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Track Topics include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Modern App Development
- Mobile Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW!
ONLY \$595

SPACE IS LIMITED



Register by December 16 and Save \$500!*

Use promo code VSLNOV2 Scan the QR code to register or for more event details. *SAVINGS BASED ON 5 DAY PACKAGES ONLY.

ROCK YOUR CODE TOUR 2017

LAS VEGAS



MARCH
13-17

AUSTIN



MAY
15-18

WASH. DC



JUNE
12-15

REDMOND



AUG
14-18

CHICAGO



SEPT
18-21

ANAHEIM



OCT
16-19

ORLANDO



NOV
13-17

CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search "VSLive"



linkedin.com – Join the
"Visual Studio Live" group!

vslive.com/lasvegas



Brain Droppings

I keep having small ideas pop into my head, ideas that can't fill an entire column, even one this short. So herewith is my next installment of brain droppings or, if you prefer, microaggressions:

As I'm teaching Annabelle to drive (and how the hell did that happen?), I wonder if I belong to the last generation of parents ever to do this. Will self-driving cars take over by the time Annabelle's kids turn 16? Will she then not teach her kids to drive, as my parents never taught me to ride a horse?

Reversing that thought, as I see my aging parents deciding when to give up driving (and how the hell did that happen?), I wonder if I belong to the last generation of adult children ever to worry about this. Will self-driving cars take over before Annabelle has to decide when to take away my keys?

Have smartphones become
so much a part of us that we
experience phantom sensations
in their absence, as an amputee
experiences a phantom limb?

Self-driving cars will finally settle the controversy of automatic transmission versus stick shift. What will I then do without that analogy to teach UX? Because if there's one thing I'd bet my house on, it's that software will still be sucking, because the developers who write it don't understand their users and, therefore, will design powerful but complex interfaces that please their geeky selves.

My daughter Lucy's phone died, and the only one I had lying around to give her was a developer edition Windows Phone 8. The girls in her eighth grade class crowded around to see it, fascinated by the Live Tiles animation (don't get me started, see my May 2013 column at msdn.com/magazine/dn198249). They seemed to regard it as doctors today regard a case of measles: Everyone wanted to look at it, because they'd heard so much about it, but never actually seen one. But, also like a case of measles, no one else wanted one of her own.

I don't like to be ruled by noises, so I often set my phone to

vibrate, even if I'm not in a space that requires quiet. I've been noticing lately that I sometimes feel a vibration in my leg under my phone pocket, even if it isn't ringing, or sometimes even if I don't have it in my pocket at all. Have smartphones become so much a part of us that we experience phantom sensations in their absence, as an amputee experiences a phantom limb?

A lot of .NET guys are feeling abandoned out there today. They see the excitement around cross-platform, or mobile devices, or the cloud. And they feel like an old, once-beloved dog, relegated to a crate, seeing the family cooing over a new puppy. But at Build 2014, Anders Hejlsberg said, "Microsoft is going all in on .NET." The components are finally rolling out of Redmond, ready for application programmers (see Julia Luisón's First Word column in this issue). I'll probably be rewriting my Harvard Extension School class next fall to cover ".NET Today." What Microsoft now needs is an outreach effort to help the old dogs learn these new tricks. I think they're smart enough, don't you? (And if Microsoft is looking for an ambassador, I'm available. Tell them you want me, OK?)

I swear I'm going to lambaste the next person who uses the term "herding cats" to describe the difficulty of managing chaotic people and situations. Herding cats is a lead pipe cinch, as I learned from Simba and her mates. All you have to do is open a can of tuna fish and walk while holding it at knee level. What would be the geek equivalent of this?

The Flying Wallendas, that famous tightrope-walking family, are appearing at a fair near me. I'm definitely going to see them. Their seven-person pyramid—four on the wire, two on their shoulders, one more in a chair on *their* shoulders—is unbelievable. Watching it on video (bit.ly/2cxk9HU) is nothing compared with experiencing it live. Our work in the software business is very abstract—queues, stacks, clouds, virtualization. Theirs is as real as it gets: a half-inch steel rope and a balance pole versus gravity. It's elemental, primal, concrete, *real*; in a way we geeks sometimes forget. They earn their standing ovations. The world will be a poorer place when they are gone from it. ■

DAVID S. PLATT teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

Manipulating Files?

View, annotate, compare, convert, assemble, sign and share over **50 types of documents on the web.**

GroupDocs.Viewer
GroupDocs.Annotation
GroupDocs.Conversion
GroupDocs.Comparison
GroupDocs.Signature
GroupDocs.Assembly
GroupDocs.Metadata
GroupDocs.Search



[Get started now](#)



.NET Libraries



Java Libraries



Cloud APIs



Cloud Apps

Contact Us:

US: +1 903 306 1676

EU: +44 141 628 8900

AU: +61 2 8006 6987

sales@groupdocs.com

Visit us at www.groupdocs.com



USE BUSINESS DASHBOARDS TO SUMMARIZE A BILLION RECORDS IN SECONDS

- Process and display massive amounts of data using commodity hardware.
- Use the Syncfusion Big Data Platform while running Apache Spark.
- Execute queries using Spark SQL.
- Consolidate metrics for easy review.

Download this white paper today!

www.syncfusion.com/dashboardwp

